تصميم برامج الحاسبات
(الخوارزميات والمخططات التدفقية)

1- مقدمة :

إن الحاسب الآلي كل متكامل عتاد وبرمجيات فبدون أي جزء لا يمكن أن يعمل ويحقق الغاية المرجوة منه ، وبالتالي فإن البرمجيات لا تقل أهمية عن العتاد وتلقى البرمجيات اهتماماً كبيراً في العالم المعلوماتي المعاصر وتشهد سوقها منافسة قوية .

وتصميم البرمجيات ينطلق من الخوارزميات والمخططات التدفقية حيث يشكلان بداية الطريق لخلق أي برنامج ، ولننطلق نحن لإيضاح هذه المفاهيم وكيفية التعامل معها ولكن لنستعرض أولاً خطوات حل مسألة باستخدام الحاسب .

2- ما هي المراحل التي نمر بها لحل مسألة ؟

نمر بخمس مراحل لحل مسألة وهي :

2-1- تعريف وتحليل المسألة :

تعريف المسألة يعني دقة في التعبير في تطبيق المسألة بحيث تصبح مفهومة بصورة واضحة دون لبث فيه لجميع من يعمل ضمن الاختصاص الذي تتناوله المسألة ، أما تحليلها ووضع الحل فهنا لب القضية وقد نعاني صعوبات في ذلك حيث يمكن أن نستخدم كثير من القوانين والطرق الرياضية المناسبة للحل وقد نضطر لتطويرها لتتناسب مع المسألة المدروسة وفي هذه الخطوة يجب تحديد ما يلي :

1- طبيعة الخرج (OUTPUT) (النتائج) وتنظيمها .

2- الدخل(INPUT) (المعطيات أو المعلومات) وتحديد نوعها وتنظيم إدخالها .

3- طرق الحل المناسبة وتقييمها بما يتلاءم مع طريقة تنفيذها وفي ضوء ذلك نختار الأفضل.

2-2- وضع الحل التخطيطي (سرد خطوات وبياني) :

نقوم هنا بالتعبير عن الحل أو الحلول التي استنتجت سابقاً على شكل خطوات متسلسلة ومترابطة منطقياً ودقيقة الوصف للوصول إلى الحل وهي ما تدعى بالخوارزمية وقيامنا بعد ذلك بوضع هذه الخطوات ضمن مخطط بياني مستخدمين مجموعة من الأشكال الاصطلاحية

والرموز نكون قد حصلنا على المخطط التدفقي لحل المسألة ويدعى أيضاً مخطط سير العمليات أو المخطط النهجي .

2-3- كتابة الكود البرمجي :

ليتمكن الحاسب من فهم هذا الحل يجب تحويله إلى لغة يفهمها وبالتالي يتم تحويل الحل التخطيطي إلى كود برمجي مكتوب بإحدى لغات البرمجة المعروفة ويسمى عندئذ بالبرنامج المصدر .

2-4- ترجمة البرنامج المصدري :

يتم ذلك بإدخال البرنامج إلى الحاسب وترجمته إلى لغة الآلة بوساطة برنامج الترجمة الخاص بلغة البرمجة المستخدمة وذلك في حال عدم وجود أخطاء في البرنامج المصدر وتمر عملية الترجمة بالمراحل التالية :

1- مرحلة التحليل المعجمي : يتم فيها مطابقة مفردات برنامج المصدر والعلاقات والأسماء مع تلك المسموح بها في لغة البرمجة المستخدمة واكتشاف الأخطاء فيها ، إن وجدت .

2- مرحلة التحليل اللغوي والنحوي : يتم فيها مطابقة تعليمات برنامج المصدر مع القواعد اللغوية للغة المستخدمة ، واكتشاف الأخطاء فيه إن وجدت .

3- مرحلة ترجمة البرنامج إلى لغة الآلة : يتم تحويل البرنامج المصدر إلى برنامج بلغة التجميع ونحصل نحن على البرنامج الهدف الذي نستطيع تنفيذه .

2-5- تنفيذ البرنامج وتجربته :

يتم تجربة البرنامج الهدف الذي حصلنا عليه للتأكد من صحته منطقياً ، وذلك باستخدام عينة من البيانات الاختبارية فإذا ثبت صحتها نكون قد حصلنا على البرنامج المطلوب ، بأفضل صورة له وجاهز للتطبيق العملي على بيانات حقيقية واستثماره .

3- ما هو مفهوم الخوارزمية أو الألغوريتم ؟

إن التعريف البسيط لكلمة خوارزمية (Algorithm) إنها طريقة أو خطة أو قاعدة للوصول اعتباراً من معطيات إلى نتائج ، ونستطيع صياغة تعريف آخر أكثر دقة كالتالي هي عبارة عن مجموعة من الخطوات المتسلسلة التي تصف بصورة مضبوطة وبدون أي غموض جميع الخطوات الرياضية والمنطقية اللازمة لحل مسألة ما ، وقد تطور هذا المفهوم وأصبح يعني طريقة أو خطة شاملة وعامة لحل مسألة ما و، تقوم بوصف كافة الخطوات بشكل مفصل ونقول عن طريقة حل مسألة بأنها خوارزمية إذا اتصفت بما يلي :

1- تحقق إمكانية وصف الطريقة المتبعة في حل المسألة بعدد من الخطوات التي تحوي تعاليم أو أوامر بشكل مفصل وصريح توضح فيها ما يجب عمله في كل خطوة بدون التباس فيه .

2- تحوي على عدد محدد من الخطوات توصلنا إلى النتائج انطلاقاً من المعطيات المتوفرة ، فوجود عدد لانهائي من الحلول لطريقة ما ، لا يمكننا من اعتبارها خوارزمية حل .

3- يجب أن تتضمن الخوارزمية جميع الشروط والاعتبارات في حل المسألة مهما اختلفت المعطيات التي تتناولها المسألة المطروحة .

ومما تجدر الإشارة إليه أن أي عمل نقوم به في حياتنا لإنجازه بالشكل الصحيح يخضع لخطة عمل محددة أي خوارزمية كما المسائل العلمية وللإيضاح نتناول مثالين أحدهما حياتي والآخر رياضي :

مثال /1/ :

ما هي الخطوات (الخوارزمية) التي نتبعها لتناول وجبة في مطعم ؟

الحل :

1- البداية .

2- الذهاب إلى المطعم .

3- اختيار مكان الجلوس .

4- طلب الوجبة.

5- تناول الوجبة .

6- استلام الفاتورة .

7- دفع الفاتورة .

8- مغادرة المطعم .

9- النهاية .

مثال /2/ :

ما هي خوارزمية حل معادلة من الدرجة الأولى من الشكل ($a + b\,x = c$) باعتبار a,b,c,x أعداد صحيحة وسيتم دراسة هذا المثال بشيء من التفصيل لتوضيح مفهوم الخوارزمية .

نلاحظ أن يمكن وضع المعادلة بالشكل : $x = \frac{c-a}{b}$

وعلى ذلك فخطوات الحل تكون التالية:

1- اطرح a من c وسمّ هذه القيمة m

2- قسم m على b وسمّ هذه القيمة x

3- اكتب قيمة x

إن هذه الخوارزمية ليست دقيقة ولا تراعي كل شروط الحل وبالتالي تعطي نتائج خاطئة في حالتين :

1- عندما b تساوي الصفر : فإذا كانت a =c فإن x يمكن أن تأخذ أي قيمة وإذا كانت a≠ c أي لا توجد أية قيمة لـ x .

2- إذا لم تعط نتيجة قسمة m على b عدداً صحيحاً فعندها لا يوجد أي عدد صحيح يحقق المعادلة السابقة .

ولكي تصبح الطريقة خوارزمية دقيقة يجب أن تراعي الحالات الخاصة وتحقق شروط الخوارزمية وعلى ذلك نستطيع كتابة الحل الصحيح بالشكل التالي :

1- إذا كانت b = 0 وكانت a = c فنفذ الخطوة السادسة من هذه الخوارزمية و إلا فتابع إلى الخطوة الثانية .

2- إذا كانت b = 0 وكانت a ≠ c فنفذ الخطوة الخامسة من هذه الخوارزمية و إلا فتابع إلى الخطوة الثالثة .

3- اطرح a من c وسميها m .

4- قسّم m على b وإذا كان هناك باقي بنتيجة القسمة فنفذ الخطوة الخامسة و إلا فاطبع النتيجة وتوقف .

5- اكتب " لا يوجد عدد صحيح يحقق المعادلة " ثم توقف .

6- اكتب " أي عدد صحيح يحقق المعادلة " ثم توقف .

وبالتالي نجد إن هذه الخوارزمية دقيقة وعامة وتحوي عدد محدد من الخطوات ومطلوب اتخاذ قرارات وإجراء مقارنات وإسناد قيم بمعنى وجود مفاهيم رياضية بسيطة يجب على قارىء الخوارزمية إدراكها إضافة لإدراك عملية الانتقال أو القفز من خطوة إلى أخرى وبالنهاية التوقف .

3-1-المتحولات :

المتحولات (Variables) عبارة عن مقادير تأخذ قيماً مختلفة وتخزن قيم المتحول في خلية من ذاكرة الحاسب وهذا يعني أننا عند تسمية متحول فإن الحاسب يخصص له خلية من ذاكرته معرفة باسمه وأن أية قيمة يأخذها هذا المتحول تخزن في خلية الذاكرة المعنونة باسمه وتزول القيمة السابقة في الخلية عند إسناد قيمة جديدة لها ونستطيع استخدام هذه القيمة بذكر

اسم المتحول ويمكننا تسمية هذه المتحولات بأي اسم نريده مثلاً حرف (X , A , B , ...) أو أكثر من حرف أو حتى كلمة والأفضل أن تحمل معنى مثل (VALUE , SUM , ...)

ونصادف ثلاث أنواع من المتحولات :

1- المتحولات العددية .

2- المتحولات المحرفية .

3- المتحولات المنطقية .

تأخذ المتحولات العددية قيماً رقمية ويمكننا إجراء العمليات الحسابية من جمع وطرح وغيره عليها أما المحرفية فلا نستطيع إجراء العمليات الحسابية عليها إنما هي أسماء أو معطيات تصنيفية وحتى لو أشرنا إليها بأرقام فهي لا تخضع للعمليات الحسابية ، وبالنسبة للمتحولات المنطقية فهي تأخذ قيمتين "صح " (TRUE) و"خطأ" (FALSE) ونحتاجها في الاختيارات و الاختبارات وتطبق عليها عمليات الجبر البولي (..... ,AND , OR) وقد تعرضنا لفكرة عن المتحول كونه سيمر استخدامها في الخوارزميات والمخطط التدفقية وهي لا تخضع لقواعد هنا بعكس ما نجده في متحولات لغات البرمجة التي تخضع لقواعد لغة البرمجة المتعامل معها .

3-2- ما هي التعاليم والأوامر الرئيسية التي يتقبلها الحاسب ؟

لا يستطيع الحاسب أن يحل مسألة ما إلا إذا لقناه الحل بمنطق يتقبله ، و يعني هذا تلقينه الحل وفق تعليمات وأوامر ومحاكمات يستطيع فهمها والتعامل معها والتعاليم والأوامر الرئيسية التي يتقبلها هي :

1- قراءة عدد أو اسم وحفظه في ذاكرته وهذا يتم في خلية محددة ومعنونة من الذاكرة نستطيع أن نغير محتوى الخلية كما نريد في البرنامج ، إلا أن عنوان الخلية يبقى ثابتاً .

2-طبع عدد أو اسم موجود في خلية محددة من الذاكرة ويعني نقل المعلومات من الخلية المحددة إلى عنصر من عناصر الخرج .

3-القيام بالعمليات الحسابية على أن يتم تحديد نوع العملية (جمع ، طرح ، .....) مع تخزين النتائج الانتقالية والنهائية في خلايا محددة من الذاكرة .

4-التوقف عن تنفيذ الأوامر والانتقال أو القفز من أمر إلى آخر في البرنامج .

5-القدرة على المحاكمة أي تقبل سؤال مطروح بشكل صريح بالبرنامج بشأن المعطيات أو إحدى النتائج التي توصل إليها على أن تكون الإجابة مقتصرة على (نعم أو لا) مع إيضاح ما يجب فعله في كل حالة من الحالتين .

3-3- ما هي الخوارزميات المبرمجة ؟

يمكننا أن نقول عن خوارزمية أنها مبرمجة (قابلة للبرمجة) إذا كنا قد وضعنا خطواتها بشكل مفصل ومنظم وفق منطق مترابط ويتقبله الحاسب وينسجم مع تكوينه أي

بمعنى آخر يجب وضع خطوات العمل في الخوارزمية المبرمجة باستخدام التعاليم والأوامر التي يتقبلها الحاسب وعند ذلك نستطيع تحويلها إلى برنامج بإحدى لغات البرمجة يلقن به الحاسب بسهولة .

4- ما هي المخططات التدفقية ؟

جاءت المخططات التدفقية (Flowcharts) كضرورة لتسهيل عمل المبرمج عندما تتعقد الخوارزمية خاصة أي تزداد خطواتها والمقارنات والمحاكمات فيها فالمخطط التدفقي هو تمثيل رسومي للخوارزمية الذي يعطينا تمثيلاً جيداً لها ويستخدم في هذه المخططات رموز وأشكال هندسية متنوعة لها دلالات محددة .

4-1- كيف يتم كتابة المخططات التدفقية ؟

يتطلب كتابة مخطط تدفقي ، يحوي عمليات حسابية وشرط ومحاكمات تمريناً طويلاً وعلى المبتدىء أن يضع إمكاناته وقدراته لإيجاد الخطوات المناسبة التي تضمن خوارزمية صحيحة ومخطط تدفقي صحيح ، حقيقة الأمر أنه لا توجد طريقة عامة متبعة لكتابة خوارزمية مبرمجة و صياغة مخطط تدفقي ، ولذلك يجب دراسة كل مسألة على حدا ويمكن وضع عدة خوارزميات لمسألة واحدة نتوصل فيها لنفس النتائج ويفيد التمرين والخبرة في ذلك كثيراً .

4-2- ما هي أهم فوائد استخدام المخططات التدفقية ؟

1- تساعد المبرمج على الإحاطة بالمسألة المراد حلها بشكل كامل والسيطرة على كل أجزائها بحيث يستفاد منها في اكتشاف الأخطاء المنطقية .

2- يصعب على المبرمج متابعة التفرعات الكثيرة التي تظهر في بعض البرامج بدون المخططات التدفقية .

3- تساعد المبرمج عند العمل على تعديل برنامج ما عند النظر إلى المخططات التدفقية فبنظرة سريعة على المخطط ندرك ماهية المسألة وإمكانية التعديل .

4- تعتبر المخططات التدفقية لحل مسألة معينة مرجعاً هاماً يجب الاحتفاظ به للعودة إليه عند الحاجة للتعديل أو الاستخدام في مسائل مشابهة .

4-3- ما هي الرموز المستخدمة في تمثيل المخططات التدفقية ؟

سنقوم باستعراضها وإيضاحها حسب ما هو معتمد في المعهد الوطني الأمريكي (AINSI) :

1- الإطار المستطيل المنتهي بنصفي دائرة :

يستخدم هذا الإطار للدلالة على نقطة بداية المخطط التدفقي أو نهايتها فنضع فيه إما كلمة البداية أو (START) في أول المخطط وكلمة النهاية أو توقف (STOP/END) عند نهاية المخطط ويجب أن يحوي المخطط على إطار واحد للبداية ولكنه قد يحوي على أكثر من إطار توقف في عدة أماكن من المخطط .

2- الإطار المتوازي الأضلاع :

يستخدم للدلالة على قراءة المعطيات وطباعتها أي لعمليات الدخل والخرج وضمنه نعين أسماء وعناوين الخلايا التي تخزن فيها المعطيات عند تنفيذ الخوارزمية وكذلك الخلايا المطلوب طباعتها فمثلاً (READ x,y) ويعني أنه عند تنفيذ الأمر يجب قراءة متحولين (عدد أو اسم) وتخزينهما في خليتين معنونتين بـ (x,y) .

3- الإطار المستطيل :

يستخدم لتحديد العمليات الحسابية وبيان الخلايا التي تخزن فيها نتائج العمليات الحسابية وهي ما تدعى بالأوامر أو التعاليم الحسابية في المخطط التدفقي فمثلاً (x = y + 5) ويعني إضافة محتوى الخلية المعنونة y إلى العدد 5 وتخزين الناتج في الخلية المعنونة x .

4- الإطار المعين :

يستخدم لوضع التساؤلات والاختبارات في المخطط التدفقي فعندما نكتب ضمن إطار معين x= y فهذا يعني هل محتوى x يساوي محتوى y ويجب أ يتفرع من إطار المعين مسارات قد تكون اثنان أو ثلاثة حسب حالة الاختبار ففي حالتنا السابقة مسارين لأن جواب السؤال سيكون إما نعم أو لا وثلاثة مسارات في حال كان السؤال x<=y فكل مسار يمثل حالة.

ملاحظة :

إن وجود التعبير x= y في إطار مستطيل يعني احفظ أو خزن محتوى الخلية المعنونة y في الخلية المعنونة x بينما وجوده في الإطار المعين يعني هل x تساوي y .

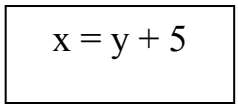
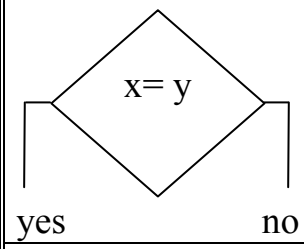
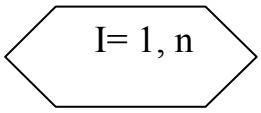
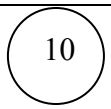
5- الإطار المستطيل المنتهي بنصفي معين :

يستخدم للتعبير عن حالات التكرار والحلقات مثلاً تكرار إعطاء قيمة لعداد I من 1 حتى n وتكتب بالشكل I= 1, n .

6- الدائرة :

تستخدم لبيان وصل منطقة من المخطط التدفقي مع منطقة ثانية ويوضع داخلها رقم أو حرف ونفس الرقم أو الحرف في المنطقة الأخرى المراد القفز إليها .

جدول يبين أشكال الرموز المستخدمة في المخططات التدفقية

اسم الرمز	شكل الرمز
الإطار المستطيل المنتهي بنصفي دائرة	
الإطار المتوازي الأضلاع	
الإطار المستطيل	
الإطار المعين	
الإطار المستطيل المنتهي بنصفي معين	
الدائرة	

ملاحظة :

يمكن كتابة الخوارزمية أو محتوى المخطط التدفقي باللغة العربية أو باللغة الإنكليزية .

4-4- ما هي الأنواع الرئيسية للمخططات التدفقية ؟

توجد أربعة أنواع رئيسية للمخططات التدفقية وهي :

1- مخططات التتابع البسيط (Simple sequential Flowchart)

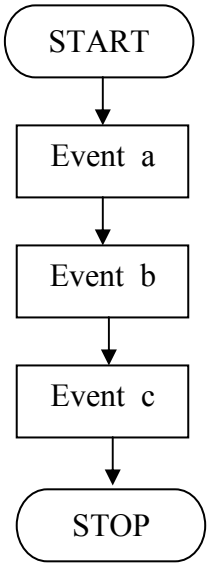
2- مخططات التفرع (Branched Flowchart)

3- مخططات التكرار البسيط (Loop Flowchart)

4- مخططات التكرارات المتداخلة (Nested – loop – Flowchart)

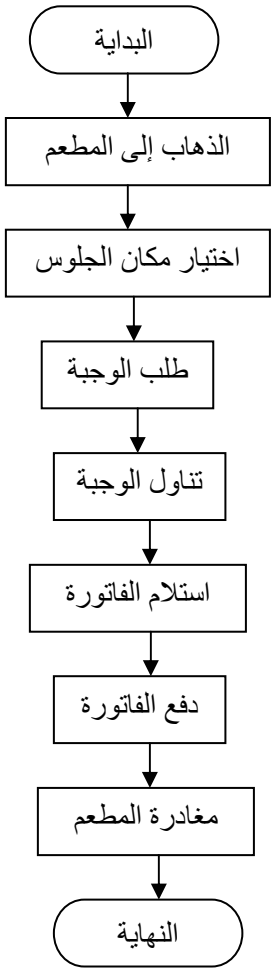
4-4-1- مخططات التتابع البسيط (Simple sequential Flowchart) :

يخلو هذا النوع من التفرعات والتكرارات أو القرارات وإنما مجموعة أوامر وأحداث متسلسلة وله الشكل العام :



مثال /3/ :

المخطط التدفقي لتناول وجبة في مطعم والواردة خوارزميته في المثال /1/



مثال /4/ :

اكتب الخوارزمية التي تمكننا من حساب محيط ومساحة دائرة نصف قطرها R وارسم المخطط التدفقي لهذه المسألة .

الخوارزمية :

1- ابدأ

2- اجعل قيمة PIE = 3.14

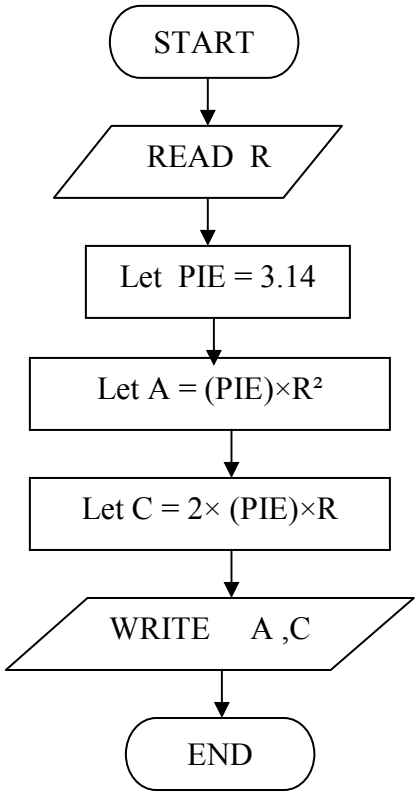
3- احسب المساحة (A) من المعادلة $A = PIE \times R^2$

4- احسب المحيط (C) من المعادلة $C = 2 \times PIE \times R$

5- اطبع قيم كل من المساحة والمحيط

6- توقف

المخطط التدفقي :



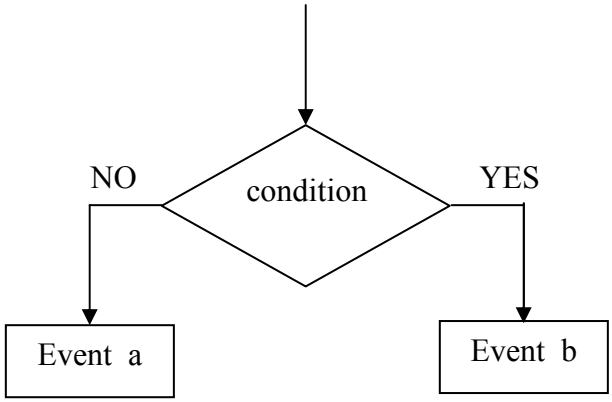
4-4-2- مخططات التفرع (Branched Flowchart) :

يتضمن هذا النوع اتخاذ القرارات أو مفاضلة بين خيارين أو أكثر وهناك أسلوبان في تنفيذ القرار

1- قرار ذو تفرعين .

2- قرار ذو ثلاث تفرعات .

والشكل العام له كما يلي :



مثال /5/ :

سنناقش مثال تناول وجبة في مطعم مع إضافة أننا نريد دفع بقشيش إضافة للفاتورة وفي هذه الحالة ستتم التغيرات التالية :

الخوارزمية :

1- البداية .

2- الذهاب إلى المطعم .

3- اختيار مكان الجلوس .

4- طلب الوجبة.

5- تناول الوجبة .

6- استلام الفاتورة .

7- هل الخدمة جيدة وأعجبتك ؟

8- إذا كان الجواب نعم تابع و إلا اذهب إلى الخطوة 10 .

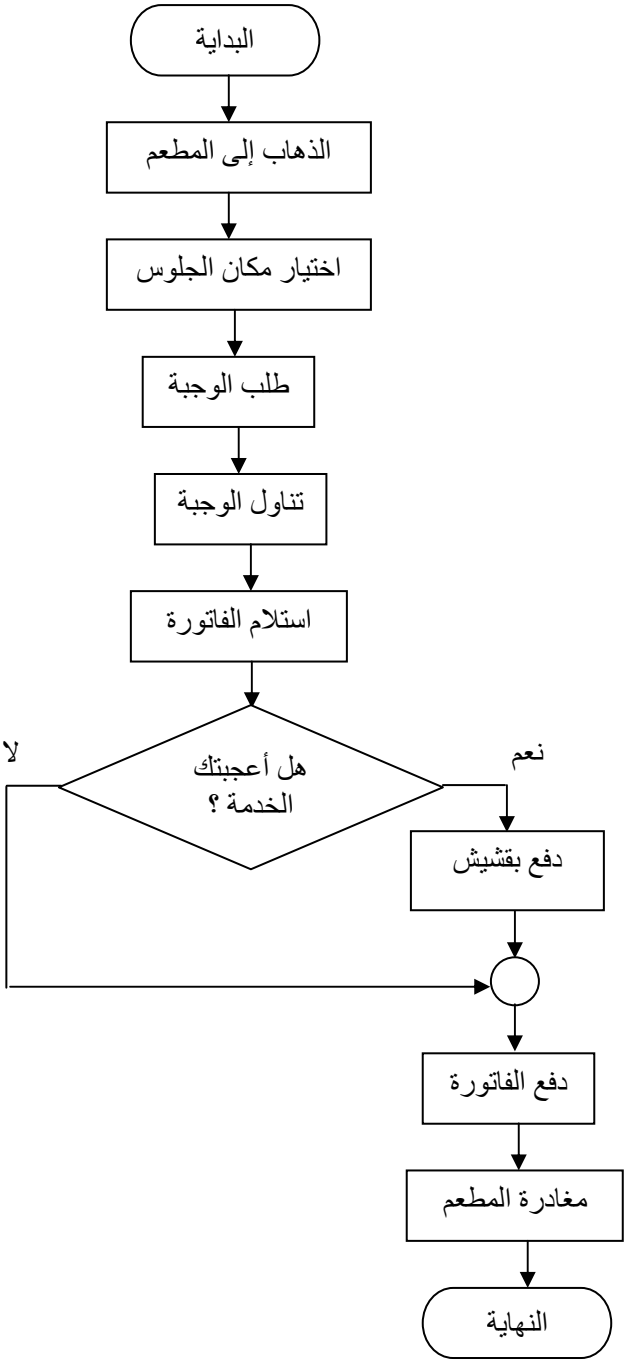
9- دفع بقشيش للعامل .

10- دفع الفاتورة .

11- مغادرة المطعم .

12- النهاية .

المخطط التدفقي :



مثال /6/ :

اكتب الخوارزمية التي تمكننا من إيجاد القيمة الأعظمية لرقمين وفق المعادلة التالية :

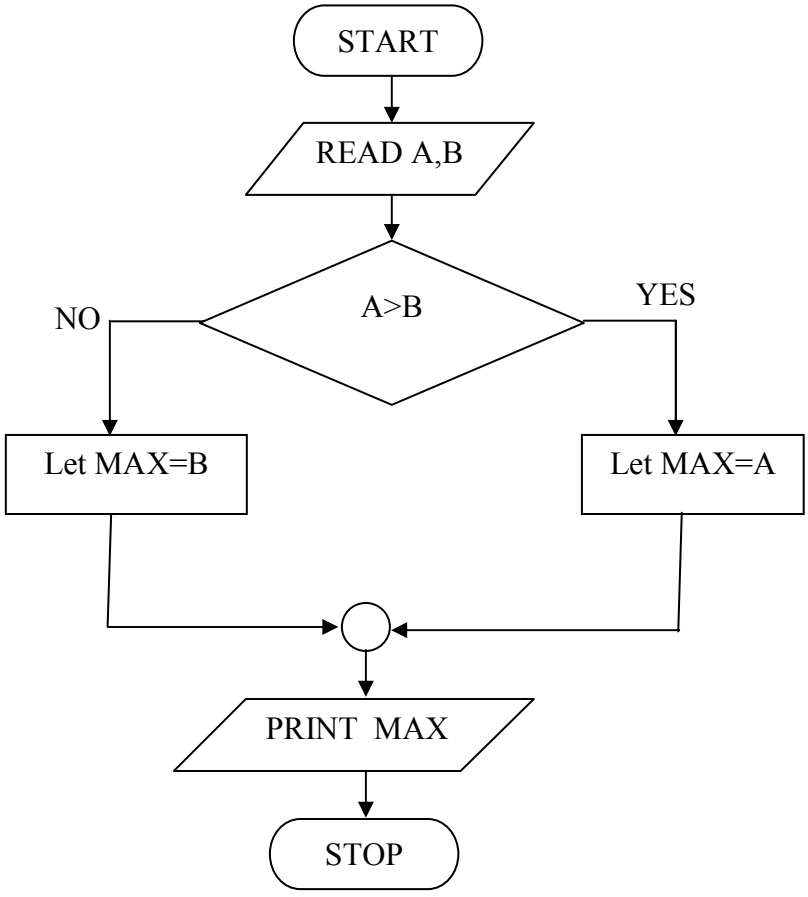
$$MAX = max (A , B)$$

ثم ارسم المخطط التدفقي الموافق .

الخوارزمية :

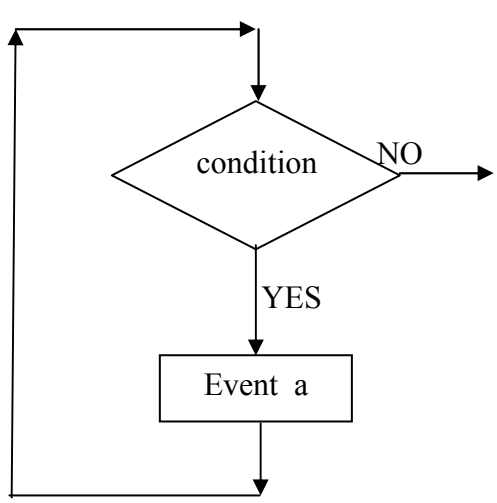
1- START
2- READ (A , B)
3- IF A>B GOTO 4 ELSE GOTO 5
4- LET MAX = A AND GOTO 6
5- LET MAX = B
6- PRINT MAX
7- STOP

المخطط التدفقي :

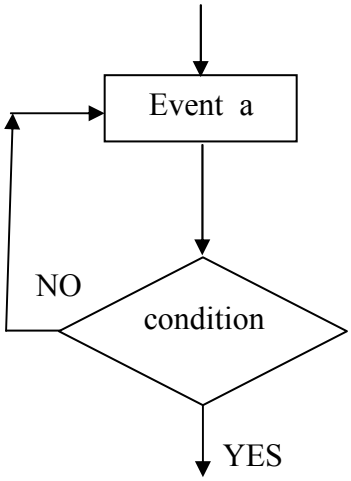


4-4-3- مخططات التكرار البسيط (Loop Flowchart) :

نحتاج لهذا النوع من المخططات لإعادة عملية أو مجموعة من العمليات في البرنامج عدداً محدداً أو غير محدود من المرات والشكل العام لها كما يلي :



يتكرر تنفيذ الحدث a عدداً من المرات طالما كان جواب الشرط نعم



يتكرر تنفيذ الحدث a إلى أن يصبح جواب الشرط نعم

مثال /7/ :

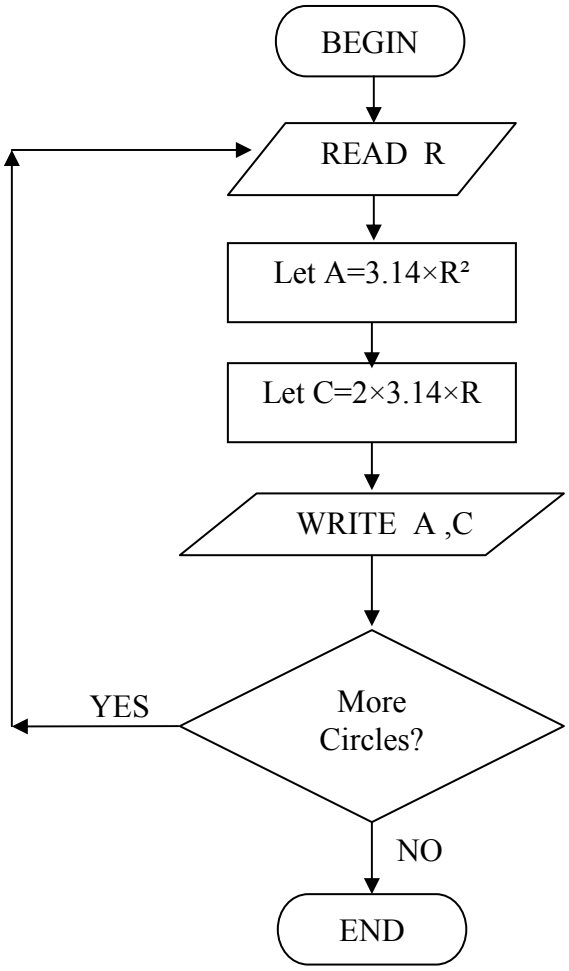
اكتب خوارزمية حساب المساحة والمحيط لمجموعة من الدوائر أنصاف أقطارها معلومة(R)

(تعديل على المثال رقم /4/)

الخوارزمية :

1- Begin
2- Read (R)
3- Let A= 3.14*R²
4- Let C= 2*3.14*R
5- Write (A ,C)
6- More Circles ? If Yes Goto (2) Else Goto (7)
7- End

المخطط التدفقي :



العداد (Counter) :

نحتاج في البرامج الحاسوبية إلى العد في كثير من الأحيان وعملية العد للإنسان طبيعية اكتسبها مع نموه خلال حياته إلا أن الحاسب يحتاج لتلقينه خطوات معينة يتبعاه ليستطيع العد ويمكن أن نحدد هذه الخطوات بما يلي :

1- اجعل العداد مساوياً للصفر

2- اجعل القيمة الجديدة للعداد تساوي القيمة القديمة له زائد واحد أي :

قيمة العداد (الجديدة) = قيمة العداد (القديمة) + 1

3- كرر الخطوات ابتداء من الخطوة (2)

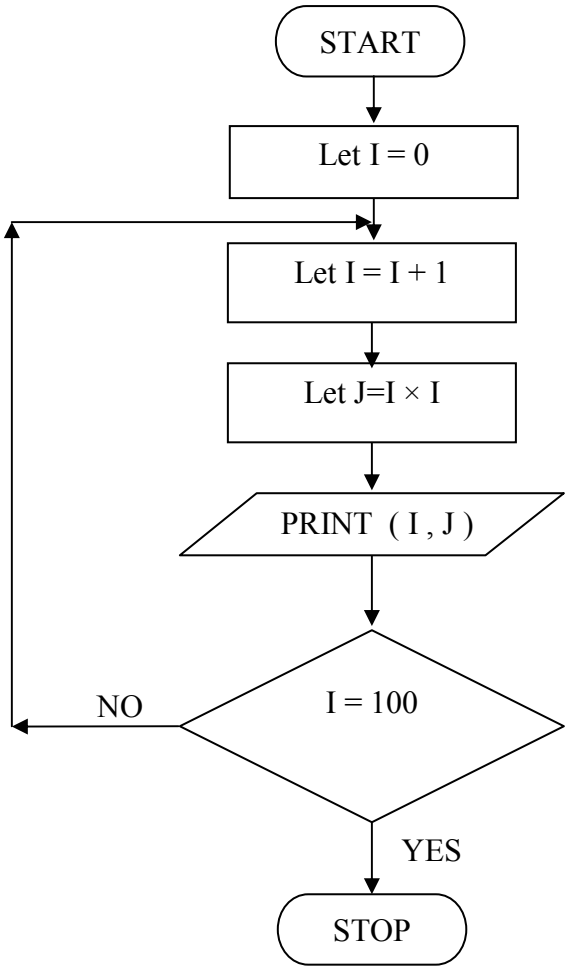
مثال /8/ :

اكتب خوارزمية طباعة الأعداد الطبيعية من 1 إلى 100 ومربعاتها وارسم المخطط التدفقي المناسب .

الخوارزمية :

1- START
2- Let I = 0
3- Let I = I + 1
4- Let J = I×I
5- PRINT (I , J)
6- If I = 100 Goto (7) Else Goto (3)
7- STOP

المخطط التدفقي :



ملاحظة :

تعتمد الزيادة في قيمة العداد على المسألة المطروحة وليس بالضرورة زيادة (1) .

المجاميع الإجمالية :

نحتاج في برامج الحاسب في كثير من الأحيان إلى جمع مجموعة كبيرة من الأعداد التي تمثل ظاهرة معينة ، فمثلاً عندما نريد أن نحسب معدل علامات طالب وكذلك الأمر في هذه الحالة يجب علينا أن نرشد الحاسب للقيام بعملية الجمع ويمكننا ذلك باستخدام متغيرين اثنين أحدهما المتغير الذي نجمعه والآخر هو الجمع الإجمالي (المجمع) ويتم ذلك وفق الخطوات التالية :

1- اجعل المجمع مساوياً للصفر

2- ادخل قيمة واحدة للمتغير

3- اجعل القيمة الجديدة للمجمع تساوي القيمة القديمة له زائد القيمة المدخلة للمتغير أي أن قيمة المجمع الجديدة = قيمة المجمع القديمة + آخر قيمة مدخلة للمتغير

4- كرر ابتداءً من الخطوة الثانية

مثال /9/ :

اكتب خوارزمية لإيجاد مجموع الأرقام من 1 إلى 20 وارسم المخطط التدفقي المناسب للمسألة المطروحة .

الخوارزمية :

1- ابدأ

2- ضع sum = 0 و I = 0

3- إذا كان جواب الشرط (I<= 20) /نعم/ اذهب إلى الخطوة 4 و إلا اذهب إلى الخطوة 6

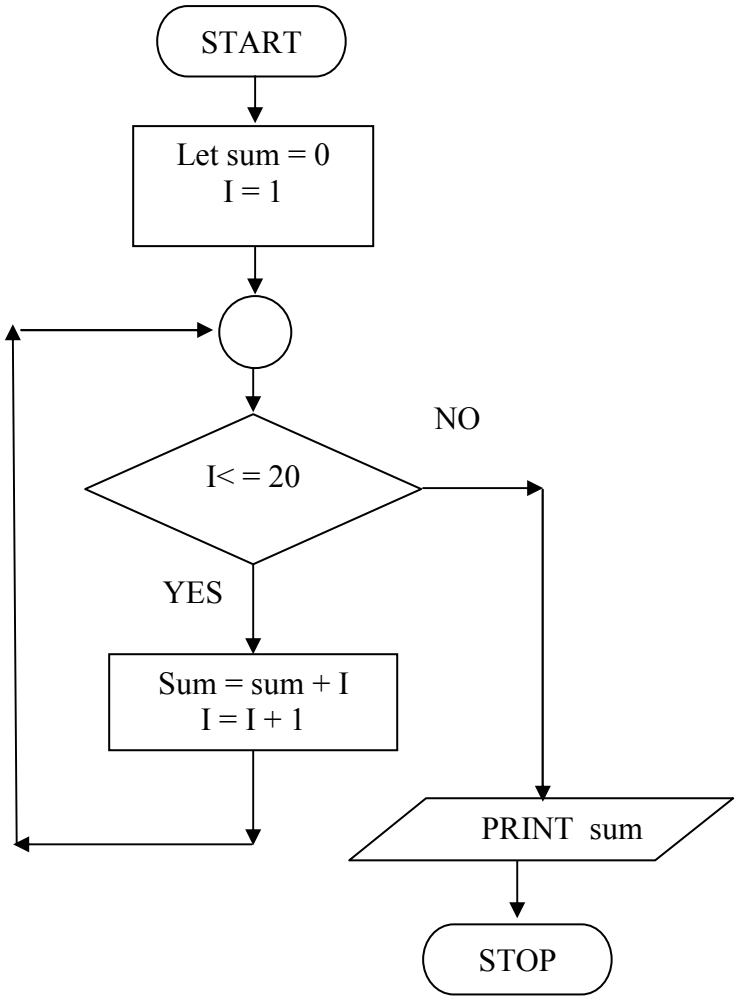
4- ضع sum = sum + I و I = I + 1

5- اذهب إلى الخطوة 3

6- اكتب المجموع sum

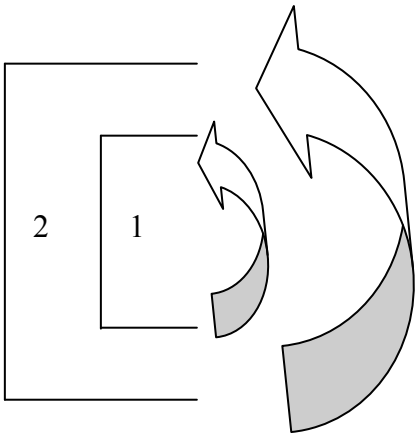
7- توقف

المخطط التدفقي :

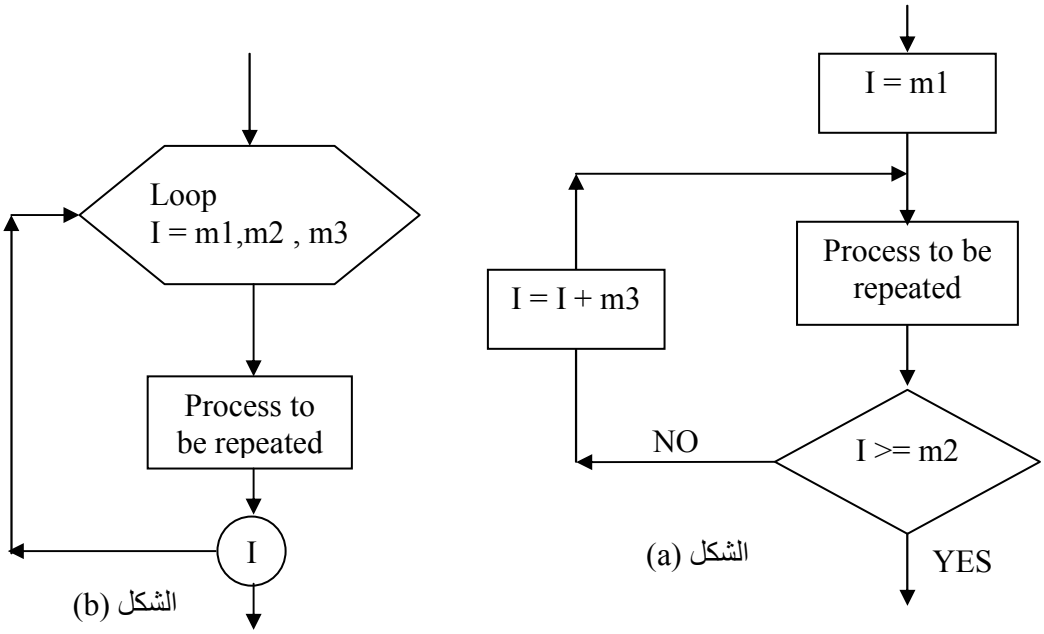


4-4-4- مخططات التكرارات المتداخلة (Nested – loop – Flowchart) :

تكون التكرارات متداخلة تماماً بحيث لا تتقاطع فإذا كان لدينا تكرارين من هذا النوع يسمى التكرار (1) تكرار داخلي بينما التكرار (2) تكرار خارجي ويتم التنسيق في عمل هذين التكرارين بحيث تكون أولوية التنفيذ للتكرار الداخلي كما في الشكل التالي :



صيغة التكرار باستخدام الشكل الاصطلاحي (الدوران) :



نلاحظ في الشكل (a) أنه لتحقيق التكرار نحتاج لما يلي :

1- العداد (I) / متغير التكرار /.

2- القيمة الأولية للعداد وتساوي m1.

3- القيمة النهائية للعداد وتساوي m2.

4- الزيادة الدورية (الزيادة عند نهاية كل تكرار) وتساوي m3.

وتكون آلية عمل هذه العناصر كما يحددها المبرمج بما يلي :

1- اجعل العداد I يبدأ بقيمة أولية مقدارها m1 .

2- أتم الإجراءات المطلوب إعادتها .

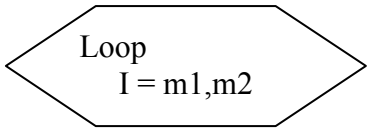
3- إذا كانت قيمة العداد I وصلت إلى القيمة النهائية m2 اذهب إلى الخطوة التالية في البرنامج وإلا فاذهب إلى الخطوة (4)

4- زد العداد I بمقدار الزيادة الدورية m3

5- عد إلى الخطوة (2)

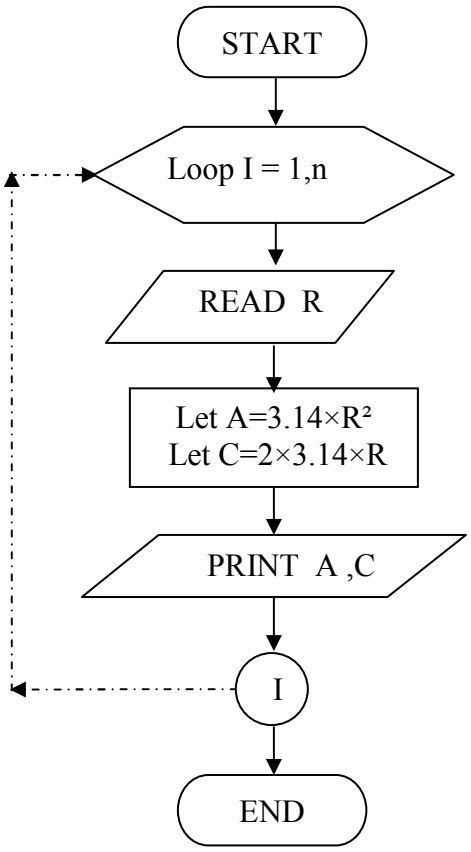
يمكننا استبدال الخطوات (1-3-4-5)في الشكل (a) بخطوة واحدة مبينة في الشكل (b) حيث ينفذها الحاسب بشكل آلي مما يؤدي إلى تسهيل عملية البرمجة واختصار عدد التعليمات وتجنب الأخطاء .

نشير إلى أن قيمة m3 تساوي /1/ دائماً ما لم تعط قيمة أخرى غير ذلك ،وفي حال عدم ذكر m3 تكون قيمتها مساوية /1/ ضمناً وتمثل كما يلي :



مثال /10/ :

اعد رسم المخطط التدفقي لإيجاد مساحة ومحيط n من الدوائر الوارد في المثال /7/ باستخدام الدوران .



5- ما هي فعّالية الخوارزمية (درجة تعقيد الخوارزمية)؟

الجدير بالذكر أنه قد يكون هناك أكثر من خوارزمية لحل مسألة واحدة وجميع الخوارزميات تؤدي إلى نفس النتيجة ولكن بطرق مختلفة وبكفاءات متفاوتة وهنا كانت الحاجة لمعرفة فعّالية الخوارزمية لاختيار الأفضل منها وتعتمد فعّالية الخوارزمية على عاملين أساسيين وهما :

1- حجم الذاكرة اللازم لتخزين هذه المعطيات وإعطاء إمكانية استخدامها .

2- الوقت اللازم لإدخال المعطيات إلى الذاكرة وكذلك الوقت المطلوب لتنفيذ الخوارزمية ويهمل العامل الأول بالقياس لأهمية العامل الثاني .

يتعلق تقييم الزمن بعدة عوامل منها حجم الدخل ويتطلب زمن من أجل التعبير عن معطيات الدخل وبالتالي فإن زمن التنفيذ تابع لـ n أي T(n) ويتعلق كذلك بنوع التعليمات والسرعة التي تنفذ فيها الآلة وهذه العوامل تعتمد على نوع الجهاز المستخدم ، ومنه فإننا لا نستطيع تحديد T(n) بشكل دقيق في واحدة الزمن الحقيقي كالثواني فهو بالتالي عدد تقريبي لعدد العمليات المنفذة وبسبب وجود عامل آخر يؤثر على الوقت وهو نوعية الشيفرة المنتجة في الحاسب نفسه ، كذلك الأمر لا يمكن تحديد T(n) لعدد العمليات المنفذة وإنما يحدد (n) عدد المرات التي يتم تنفيذ الخوارزمية فيه .

مثال /11/ :

لنكتب خوارزمية حساب متوسط مجموعة قيم ونستنتج درجة تعقيدها .

رقم الخطوة	الخوارزمية	عدد مرات التنفيذ
1	اقرأ عدد الأرقام (n)	1
2	اجعل المجموع يساوي الصفر (sum = 0)	1
3	اجعل العداد يساوي الصفر (I=0)	1
4	طالما I<=n نفذ	n+1
5	اقرأ الرقم	n
6	أضف الرقم إلى المجموع	n
7	قم بزيادة I بمقدار 1	n
8	احسب المتوسط = المجموع ÷ عدد الأرقام (sum/n)	1
المجموع		4n+5

ومنه ينتج أن الزمن اللازم لهذه الخوارزمية يعطى بالعلاقة :

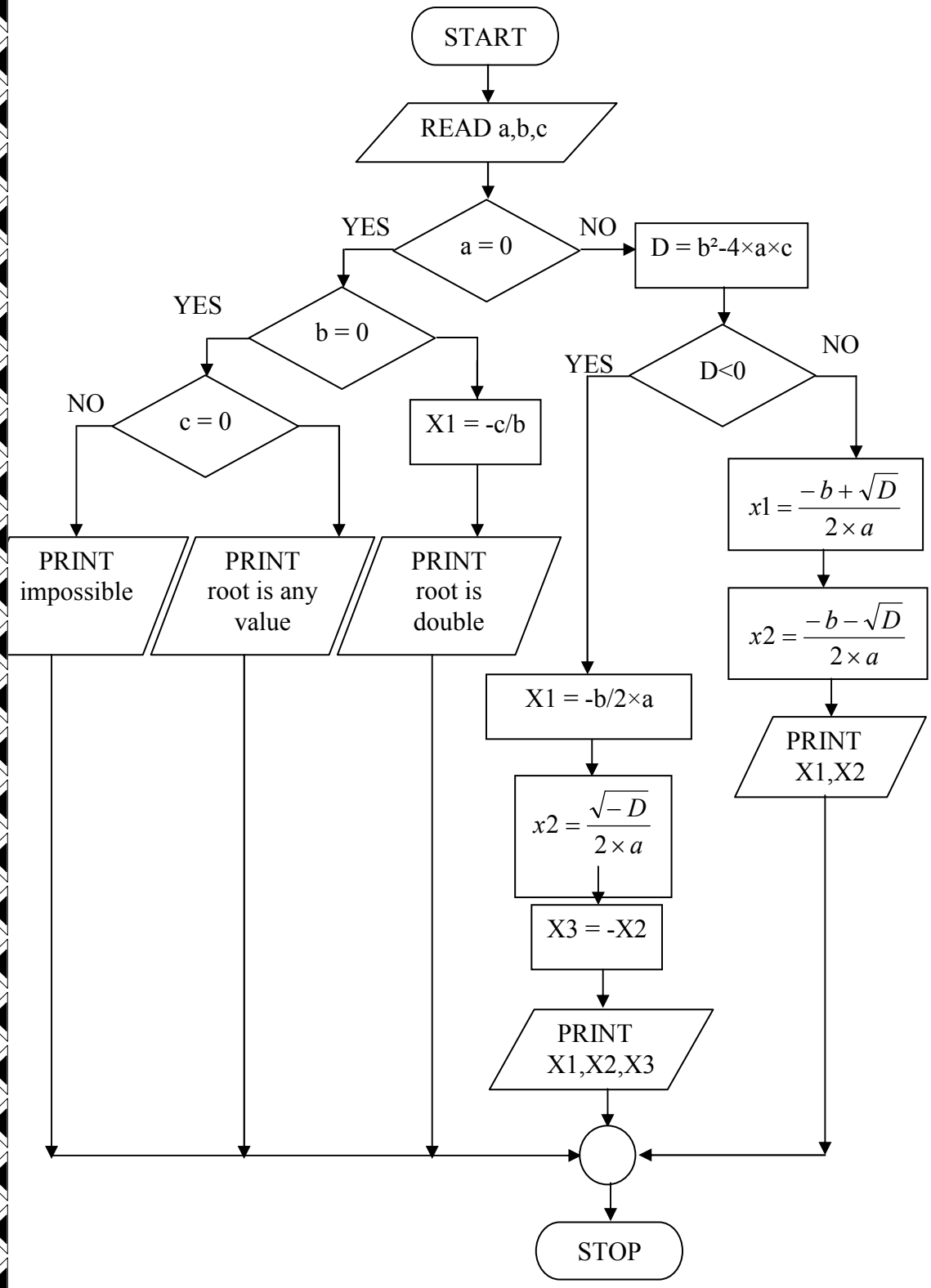
$$T(n) = 4n + 5$$

حيث n عدد قيم الدخل

نلاحظ أن العلاقة تابع درجة أولى بالنسبة لـــ n إلا أنه قد تكون الخوارزمية أكثر تعقيداً وينتج تابع يحتمل أي شكل للمعادلة كالدرجة الثانية أو الثالثة أو حتى تابع لوغاريتمي أو أسي .

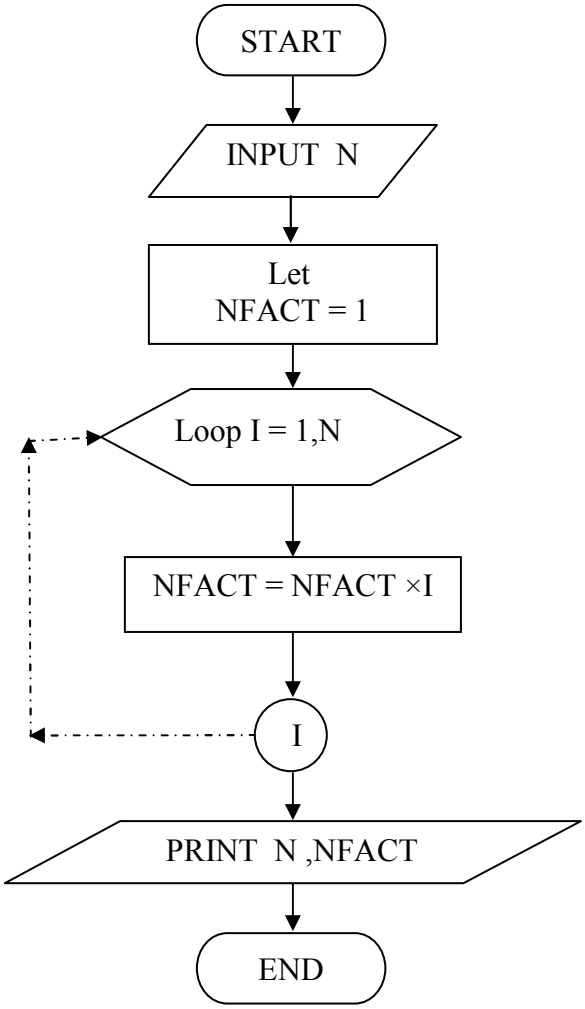
6- تطبيقات على الخوارزميات والمخططات التدفقية :

1- ارسم المخطط التدفقي لحل معادلة من الدرجة الثانية من الشكل : $a x^2 + b x + c = 0$



2- ارسم المخطط التدفقي لإيجاد العاملي (nعاملي) والتي تعطى بالعلاقة :

N ! = n(n-1)(n-2)(n-3) .....2×1



3- قامت إحدى الشركات التجارية في مدينة اللاذقية بعرض كمية من السجاد وتقدم عدد من المناقصين للشراء حسب المناقصة المعلنة وبفرض الأسعار بالليرة السورية هي التالية :

X1 , X2 , X3 , ...

وإذا اعتبرنا n عدد المتقدمين فاكتب الخوارزمية التي يتبعها الحاسب لاختيار أكبر مبلغ مقدم لشراء الصفقة وارسم المخطط التدفقي للحل .

الحل :

نلاحظ أنه لدينا مجموعة من عروض الأسعار لنعتبرها مرتبة في نسق X على الشكل التالي :

X(1),X(2),X(3), ...,X(n)

ونفترض أن XMAX يمثل أعلى عروض الأسعار فيكون :

الخوارزمية :

1- ابدأ

2- ادخل العدد الكلي للمتقدمين (n)

3- ادخل عروض الأسعار X(1),X(2),X(3), ...,X(n)

4- اجعل XMAX =X(1)

5- اجعل العداد I = 2

6- إذا كان XMAX >=X(1) اذهب إلى الخطوة 8 وإلا اذهب إلى الخطوة 7 (مقارنة)

7- اجعل XMAX = X(1)

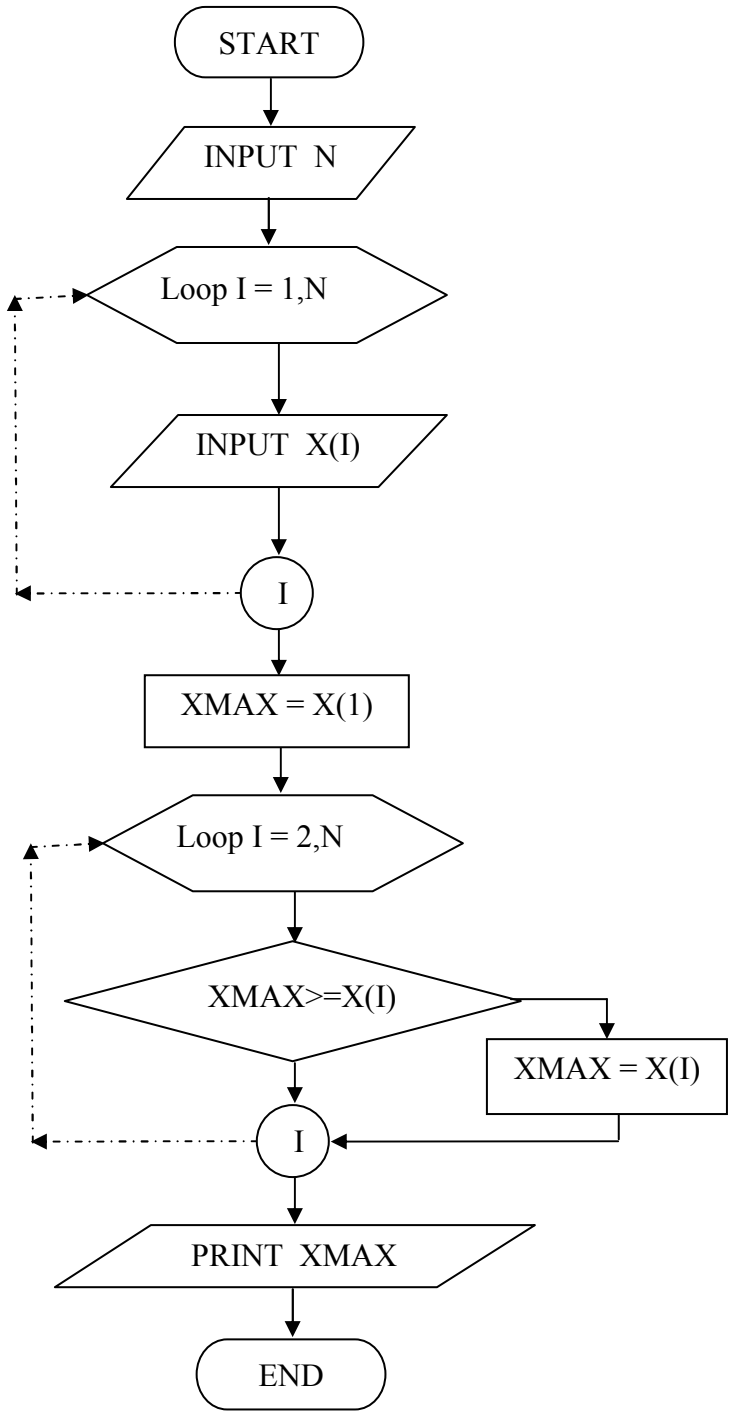
8- اجعل I = I + 1

9- إذا كان I>n اذهب إلى الخطوة 10 و إلا عد إلى الخطوة 6

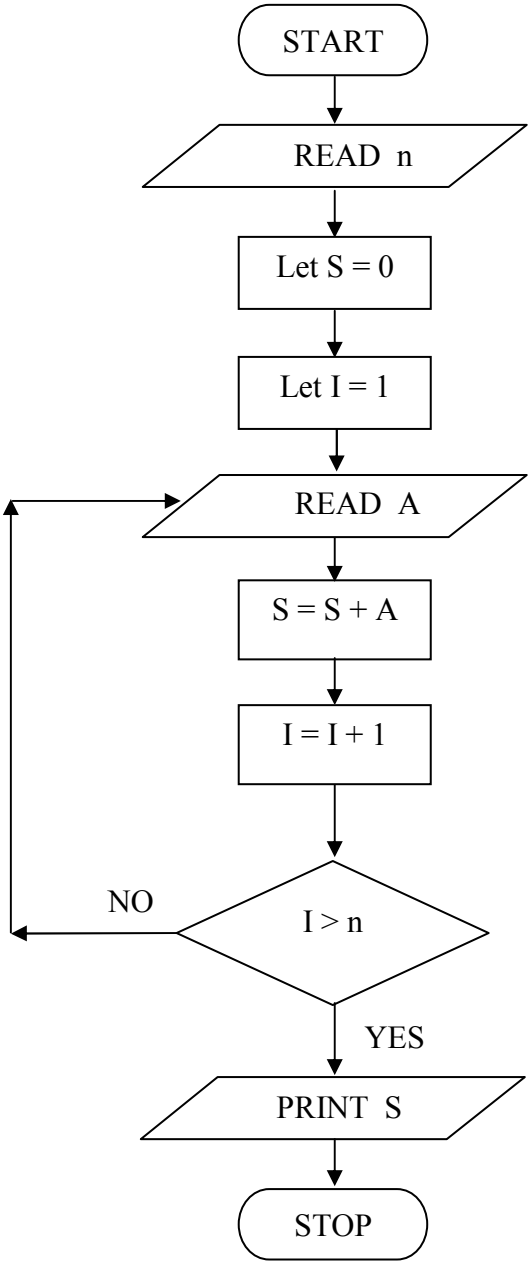
10- اطبع XMAX

11- النهاية

المخطط التدفقي :



4- ارسم المخطط التدفقي لحساب مجموع n عدد .



ملاحظة :

قمنا بكتابة الخوارزميات والمخططات التدفقية باللغة العربية تارة والإنكليزية تـــارة أخـــرى واستعمال كلمات متنوعة كلها مستخدمة في هذا المجال ليصار إلى التآلف مع هذه المفردات .

****

الفهرس

- **قسم الخوارزميات**

1. الخوارزميات العودية - Recursive Algorithms 2
2. الخوارزميات التراجعية - Backtracking Algorithms 20
3. التعقيد الزمني - Complexity 33

- **قسم بنى المعطيات**

1. بنى المعطيات الخطية 39
   1.1. القوائم المترابطة - Linked Lists 39
   1.2. المكدسات - Stack 48
   1.3. الأرتال - Queue 54
2. بنى معطيات غير خطية 60
   2.1. الأشجار - Trees 60
      2.1.1. الأشجار الثنائية - Binary Trees 60
      2.1.2. الأشجار المعممة - Generalized Trees.. 74
   2.2. البيان - Graph 77

الخوارزميات العودية - Recursive Algorithms

نقول عن برنامج جزئي (تابع / إجراء) أنه عودي عندما يستدعي نفسه (recursive call). و بالتالي يتم استدعاء هذا التابع بشكل مكرر (أي يتم تنفيذ نفس التعليمات بشكل مكرر) .
من المفيد في بعض المسائل أن نستخدم مفهوم العودية بدلاً من حلقات تكرارية, خاصة في مسائل الذكاء الصنعي و مسائل الترتيب ...
نلاحظ أن تكرار الاستدعاء العودي لا نهائي لذلك يجب بالضرورة وضع شرط توقف لإنهاء التكرار .

فائدة الخوارزميات العودية:
عندما نتعامل مع مسائل معقدة تقوم العودية بتقسيم هذه الحالة إلى حالات أبسط على التتالي حتى أن تصل إلى حالة بدائية . و من ثم تعود خطوة خطوة و تعوض القيم الناتجة إلى أن تصل إلى الحالة الأصلية فنحصل على الناتج النهائي .

السلبيات:
1- الاستدعاءات العودية تسبب ضياعاً في الوقت و تستهلك ذاكرة إضافية (لأن كل استدعاء يُعتبر تابع جديد يتم حجز متحولاته المحلية و وسطاءه من جديد في المكدس , مما يؤدي في الأخير إلى الطفحان = Stack Overflow) .
2- لا تقبل بعض اللغات البرمجية التعريف العودي للتوابع و منها مثلاً : لغة الآلة , Cobol , Fortran ...

نتيجة: يجب أن يكون عمق العودية (أي عدد تكرار الاستدعاءات العودية لنفس التابع) منتهياً و صغيراً .
ملاحظة: أحياناً فإن اللجوء إلى العودية ليس هو الحل الأمثل بل يُفضل عليه استخدام الخوارزميات التكرارية .

أنواع الخوارزميات العودية:
يمكن أن تكون التوابع ذات عودية مباشرة أو ذات عودية غير مباشرة , حيث يبين الجدول التالي الفرق بينهما :

التوابع ذات العودية المباشرة	التوابع ذات العودية غير المباشرة
هي توابع تحوي استدعاءً صريحاً لنفسها . مثال :	هي توابع تحوي استدعاءً لتوابع أخرى و التي بدورها تستدعي التابع الأب . مثال : إذا كان لدينا التابع

int A(void) {
……
A( ); // direct recursive call
……
}
	void B(void) {
	……
	A( );
	……
	}
	
	int A(void) {
	……
	B( ); // indirect recursive call
	……
	}

أمثلة لبرامج عودية :

❖ حساب العاملي لعدد – Fact :

من المعروف أن قانون العاملي يُحسب كالتالي : $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots\ldots \cdot 2 \cdot 1$
و لكن يمكن كتابته بشكل آخر : $n! = n \cdot (n-1)!$

$$(n-1) \cdot (n-2)!$$
$$(n-2) \cdot (n-3)!$$
$$\ldots\ldots\ldots\ldots$$
$$1 \cdot (0!)$$

إذاً تتكرر عملية حساب العاملي لعدد ما و حيث أن هذا العدد بدأ من القيمة n و يتناقص في كل مرة حتى أن يصل إلى الصفر , علماً أن الصفر أصغر قيمة مقبولة لحساب العاملي لها . لذلك يمكن هنا تطبيق مفهوم العودية لأننا نكرر عمليات حساب العاملي لأعداد متناقصة حتى أن نصل إلى الحالة البدائية .
لنكتب تابع حساب العملي لعدد بطرقتين (تكرارية و عودية) :

حساب العاملي لعدد تكرارياً	حساب العاملي لعدد عودياً

long fact(int n)
{ long f=1;
if ((n==1)||(n==0)) return 1;
else
{ for (int i=1;i<=n;i++)
f*=i;
return f; }
}
	long fact(int n)
	{ if ((n==1)||(n==0))
	return 1;
	else
	return n*fact(n-1);
	}

❖ حساب قيمة متتالية فيبوناتشي عند عدد ما - Fibonacci :

إن الشكل العام لمتتالية فيبوناتشي هو : $F_n = F_{n-1} + F_{n-2} \quad ; n \geq 2 \ (F_0 = 0 \ , \ F_1 = 1)$

و بالتالي لدينا مثلاً :

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	……
0	1	1	2	3	5	8	13	……

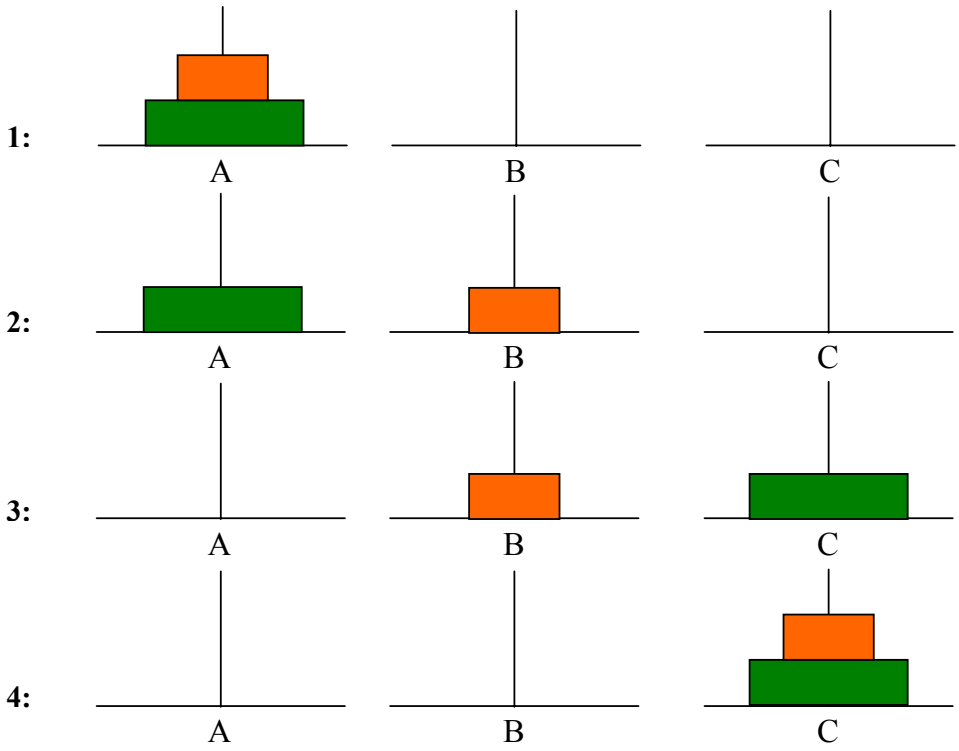
إذاً لدينا حالتين بدائيتين (0 و 1) بينما من أجل أي عدد آخر لا يمكن حساب القيمة إلا بالاعتماد على القيم المسبقة , هذا يعني أنه لدينا تكرار حساب تابع فيبوناتشي و لكن في كل مرة لأعداد أصغر حتى أن نصل إلى أبسط قيم ممكنة (أي 0 و 1) .
و يكتب التابع العودي كما يلي :

```
int fib(int n)
{ if (n==0) return 0;
  else
  if (n==1) return 1;
  else      return fib(n-1)+ fib(n-2);
}
```

❖ مسألة أبراج هانوي - Towers of Hanoi :

ليكن لدينا n قرصاً من أقطار مختلفة . و لكل قرص ثقب بحيث يمكن إدخاله إلى عامود . و المطلوب:
بناء برج هرمي من هذه الأقراص على عامود C علماً بأنها كانت متوضعة على شكل برج هرمي على عامود A مع التقيد بالشروط التالية :
- يُسمح في كل خطوة بنقل قرص واحد فقط من عامود لآخر .
- يُمنع وضع قرص أكبر على قرص أصغر .
- يمكن استخدام العامود المساعد B مرحلياً .

(كتنفيذ يُطلب فقط طباعة مراحل النقل , أي معرفة نقل أي قرص و إلى أي عامود على التتالي).
فمثلاً من أجل n = 2 لدينا :



الحل:

```
#include <iostream.h>
int n;
void hanoi(int n, char A, char B, char C);
void main()
{
  do { cout<<"Enter n= ";
       cin>>n;
   } while (n<=0);
  hanoi(n,'A','B','C');
}
/*****************************************************************************/
void hanoi(int n, char A, char B, char C)
{
  if (n==1) cout<<A<<" --> "<<C<<endl;
  else
      { hanoi(n-1,A,C,B);
        cout<<A<<" --> "<<C<<endl;
        hanoi(n-1,B,A,C);
      }
}
```

حيث يعطي تنفيذ البرنامج السابق النتائج التالية (من أجل قرص أو قرصين أو ثلاثة أقراص) :

n = 1	n = 2	n = 3
A → C	A → B A → C B → C	A → C A → B C → B A → C B → A B → C A → C

❖ تحويل التعابير الرياضية من مُصدرة إلى مُلحقة :

نستطيع كتابة التعابير الرياضية بلا أقواس باستخدام أسلوبين مختلفين : الكتابة المُصدرة (Prefix) و الكتابة المُلحقة (Postfix) . في الأسلوب المُصدر تسبق كل عملية (Operator) مباشرة عواملها (Operands) أو متحولاتها مباشرة . بينما في الأسلوب المُلحق تتبع كل عملية عواملها مباشرة .

أمثلة:

نظامي (Infix)	مصدر (Prefix)	ملحق (Postfix)
A+B	+AB	AB+
A+B*C	+A*BC	ABC*+
A*(B+C)	*A+BC	ABC+*
A*B+C	+*ABC	AB*C+
A+B*C+D-E*F	-++A*BCD*EF	ABC*+D+EF*-
(A+B)*(C+D-E)*F	**+AB-+CDEF	AB+CD+E-*F*

إن أفضل طريقة لتعريف هذين الأسلوبين برمجياً هي باستعمال العودية . لنفترض للتسهيل عدم وجود ثوابت في التعابير الرياضية , كما نفترض أيضاً أن جميع العمليات ثنائية .

المخطط الأولي للخوارزمية العودية التي تقوم بعملية التحويل من تعبير مُصدر إلى تعبير مُلحق هو كالتالي :

- إذا كان التعبير متحولاً وحيداً , فلا نقوم بأي شيء (تعبيران متكافئان في هذه الحالة).
- تحديد أول عملية حسابية في التعبير المصدر و لنسميها op .
- إيجاد المعامل الأول opnd1 للتعبير المُصدر و تحويله إلى مُلحق و تسميته post1 .
- إيجاد المعامل الثاني opnd2 للتعبير المُصدر و تحويله إلى مُلحق و تسميته post2 .
- دمج التعابير الجزئية op , opnd2 , opnd1 بالترتيب و وضع الناتج في التعبير المُلحق .

لنحدد أولاً بنى المعطيات المستخدمة :

```
const  int  strsize = 80;
struct string {
       char ch[strSize];
       int  length;
 };
```

حيث نلاحظ في تعريف سلسلة المحارف string أن ch هي سلسلة المحارف نفسها و length تمثل الطول الجاري للسلسلة .
نحتاج أيضاً إلى تعريف الإجرائيتين المساعدتين التاليتين :

- الإجرائية concat(a,b) التي تقوم بدمج السلسلتين a و b في السلسة c . فمثلاً الاستدعاء concat من أجل "abc" و "xy" سيضع في c السلسلة "abcxy" .
- الإجرائية S2= substr(S1, i , j) التي تضع في السلسلة الجزئية S2 جميع الأحرف التي عددها j من S1 اعتباراً من الموقع i في S1 (أي نسخ جزء من السلسلة S1 إلى السلسلة S2). فمثلاً الاستدعاء S2= substr("Welcome",4,2) سيضع في S2 السلسلة "om" .

أما بالنسبة للخوارزمية العودية لإجراء عملية التحويل سنسميها convert , و هي أيضاً بدورها ستعتمد على خوارزمية عودية أخرى اسمها find و التي تحسب طول معامل (أي عدد المحارف التي تؤلف المعامل) .

الحل:

```
#include <iostream.h>
#include <string.h>    //  -->  strlen(const char *s)
#include <ctype.h>     //  -->  isalpha(int c)

const int strSize=80;
struct string {
       char ch[strSize];
       int  length;
 };
string prefix , postfix;    //  prefix = input , postfix = output
char  pr[strSize];         //  users first input (we convert it to string)
int i;

string  concat(string s1,string s2);       //  merge two strings
string  substr(string s1,int i,int j);    // copy part from s1 to a new string
string  convert(string prefix);           //  convert prefix into postfix
int     find(string s,int position);     //  returns the length of an operand

/*****************************************************************************/
void main()
{
  cout<<"Enter a mathimatical sentence in prefix_case :\n";
  cin.getline(pr,strSize,'\n');     //  reads text with white spaces
  prefix.length=strlen(pr);
  for (i=0;i<prefix.length;i++)
      prefix.ch[i]=pr[i];
  postfix=convert(prefix);
  cout<<"Postfix = ";
  for (i=0;i<postfix.length;i++)
       cout<<postfix.ch[i];
}
/*****************************************************************************/
string concat(string s1,string s2)
{ string s3;
  s3.length = s1.length + s2.length;
  for (i=0;i<s1.length;i++)
      s3.ch[i]=s1.ch[i];
  int j=s1.length-1;
  for (i=0;i<s2.length;i++)
      { j++;
        s3.ch[j]=s2.ch[i];
      }
  return s3;
}
```

```
string substr(string s1,int i,int j)
{ string s2;
  s2.length=j;
  int m=0;
  for (int k=i;k<i+j;k++)
      { s2.ch[m]=s1.ch[k];
        m++;
      }
  return s2;
}
/*****************************************************************************************/
string convert(string prefix)
{ string postfix;
  char op;              // op = operator
  string opnd1,opnd2,temp;   // opnd1= first operand , opnd2= second operand
  string post1,post2,opstr; // post1= opnd1 after converting ,
                            //  post2= opnd2 after converting , opstr= op as string
  int m,n;                 //  m = length of opnd1   ,   n = length of opnd2

  if (prefix.length==1)
      if (isalpha(prefix.ch[0]))
          postfix=prefix;
      else
         { cout<<"Illegal prefix string !\n"; postfix.length=0; }
  else
      {
       op=prefix.ch[0];
       m=find(prefix,1);
       n=find(prefix,1+m);
       if (((op!='+')&&(op!='-')&&(op!='*')&&(op!='/'))||
            (m==0)||(n==0)||(m+n+1!=prefix.length))
           { cout<<"Illegal prefix string !\n"; postfix.length=0; }
       else
           {
            opnd1=substr(prefix,1,m);
            opnd2=substr(prefix,1+m,n);
            post1=convert(opnd1);
            post2=convert(opnd2);
            temp=concat(post1,post2);
            opstr.length=1;
            opstr.ch[0]=op;
            postfix=concat(temp,opstr);
           }
      }
  return postfix;
}
/*****************************************************************************************/
int find(string s,int position)
{ if (position>=s.length)   return 0;
  else
     { char first; int m,n;
       first=s.ch[position];
       if (isalpha(first))  return 1;
       else
          { m=find(s,position+1);
            n=find(s,position+1+m);
            if ((m==0)||(n==0)) return 0;
            else                return m+n+1;
          }
     }
}
```

❖ مسألة الفرز بالدمج - Merge Sort :

ليكن لدينا الجدول t , عناصره من الأعداد الصحيحة غير المرتبة , و نريد ترتيب هذه العناصر ضمن الجدول بطريقة الفرز بالدمج (Merge Sort). و تتلخص هذه الطريقة بما يلي: نقسم الجدول إلى جدولين جزئيين , و نرتب كل جدول جزئي على حدة , ثم نقوم بدمجهما في جدول مرتب واحد . عملية ترتيب كل جدول جزئي هي تطبيق لنفس الطريقة لكن عند مستوى أقل . نتابع هذه العملية وصولاً إلى جداول جزئية مؤلفة من عنصر واحد فقط , ثم نقوم بدمج كل جدولين مع المقارنة حسب الترتيب.

نحن إذاً أمام مسألة عودية .

لو أخذنا كمثال جدول مؤلف من 6 أعداد صحيحة غير مرتبة , فإن خطوات خوارزمية الفرز بالدمج كالتالي :

```
[  9   7   3   0   6   1  ]
[ 9   7   3 ]      [ 0   6   1 ]
[ 9   7 ] [ 3 ]    [ 0   6 ] [1 ]
[ 9 ] [ 7 ] [ 3 ]  [ 0 ] [ 6 ] [1 ]
[ 7   9 ] [ 3 ]    [ 0   6 ] [ 1]
[ 3   7   9 ]      [ 0   1   6 ]
[  0   1   3   6   7   9  ]
```

لحل المسألة سوف نعتمد على إجراءين , هما :

```
void merge(int low,int mid,int high,tableType t); //  لدمج جدولين جزئيين مرتبين في جدول مرتب وحيد
 void mergeSort(int low,int high,tableType t);    //  إجراء عودي يقوم بتجزئة جدول و ترتيب كل جدول جزئي على حدا و ثم يطلب دمج الجدولين الجزئيين المرتبين
```

مخطط خوارزمية الـ mergeSort يكون على الشكل التالي (حيث أن low و high هما أدلة بداية و نهاية الجدولين الجزئيين) :

طالما شرط التوقف غير محقق :

1- حساب دليل منتصف الجدول t و وضع القيمة المقابلة لهذا الدليل في المتحول mid .
2- ترتيب الجدول الجزئي t[low .. mid] بطريقة الفرز بالدمج .
3- ترتيب الجدول الجزئي t[mid+1 .. high] بطريقة الفرز بالدمج .
4- دمج الجدولين الجزئيين المرتبين في جدول وحيد مرتب t[low .. high] .

ملاحظة: نفترض أن الجدولين الجزئيين متجاورين, بمعنى أن دليل بداية الجدول الجزئي الثاني يساوي دليل نهاية الجدول الأول +1.

و سوف نستخدم جدولاً مساعداً aux لتخزين العناصر قبل نقلها إلى الجدول t .

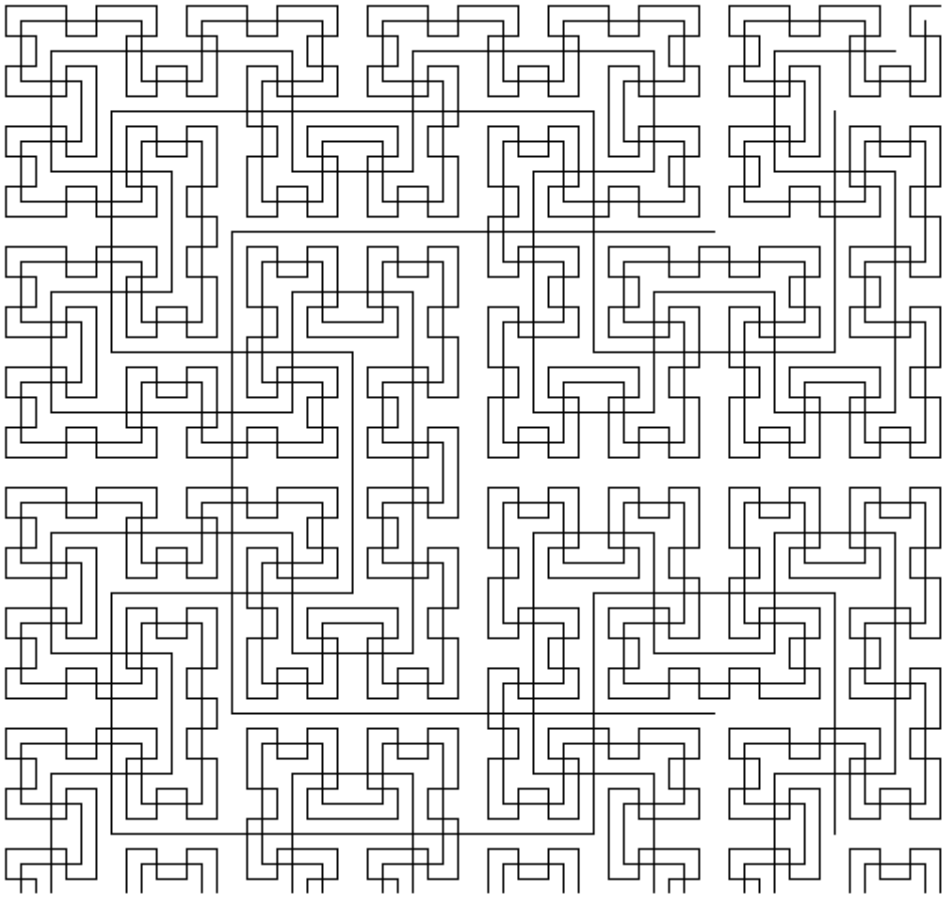
الحل:

```
#include <iostream.h>
const int n=100;
typedef int tableType[n];
tableType t,aux;        //  t= unsorted table  ,  aux= auxiliary table (temporary)
int m;           //  dimension of table

void merge(int low,int mid,int high,tableType t);
void mergeSort(int low,int high,tableType t);     //  recursive algorithm

/*****************************************************************************/
void main()
{
  do { cout<<"Enter dimension of table , dim = ";
       cin>>m;
   } while (m<=0);
  for (int i=0;i<m;i++)
      { cout<<"t["<<i<<"]= ";
        cin>>t[i];
      }
  mergeSort(0,m-1,t);
  for(i=0;i<m;i++)
      cout<<t[i]<<"  ";
}
/*****************************************************************************/
void merge(int low,int mid,int high,tableType t)
{  int i=low, j=mid+1;  //i=index in first half of t , j=index in second half of t
   int k=low;           // k=index in aux
   while ((i<=mid)&&(j<=high))
      {
        if (t[i]<t[j])
```

```
            { aux[k]=t[i];
              k++;
              i++;
            }
          else
          if (t[i]>t[j])
            { aux[k]=t[j];
              k++;
              j++;
            }
          else        //  repition of same number

            { aux[k]=t[i];      //  or:  aux[k]=t[j]
              aux[k+1]=t[i];
              k+=2;
              i++;
              j++;
            }
       }
    while (i<=mid)
       { aux[k]=t[i];
         k++;
         i++;
       }
    while (j<=high)
       { aux[k]=t[j];
         k++;
         j++;
       }
    i=low;
    while (i<=high)
       { t[i]=aux[i];
         i++;
       }
}
/*****************************************************************************/
void mergeSort(int low,int high,tableType t)
{   if (low!=high)
       { int mid=(low+high)/2;
         mergeSort(low,mid,t);
         mergeSort(mid+1,high,t);
         merge(low,mid,high,t);
       }
}
```

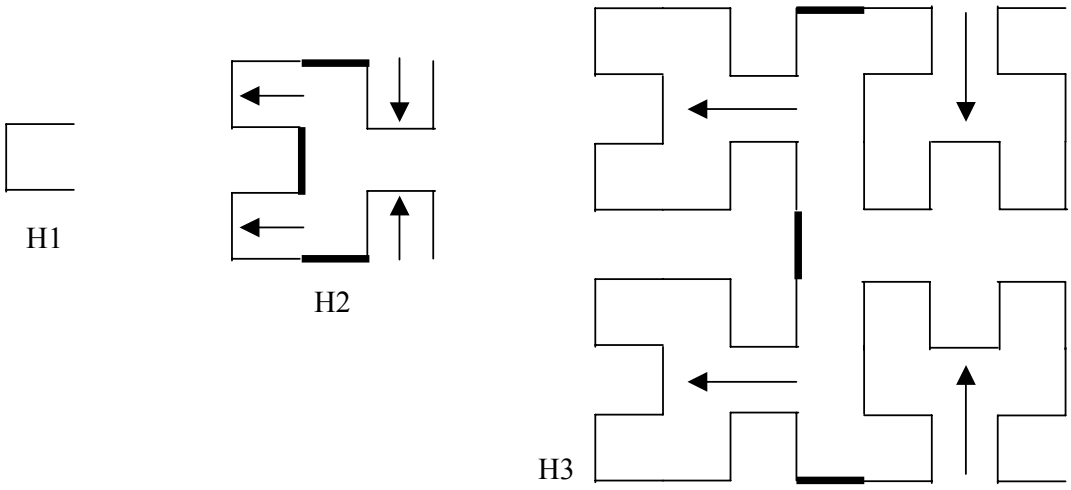
❖ منحنيات هيلبرت - Hilbert :

إن منحنيات هيلبرت مثال عن الزخرفة المتناسقة , حيث أنها عبارة عن تركيب عدة خطوط منكسرة تتبع منهجاً في الرسم.



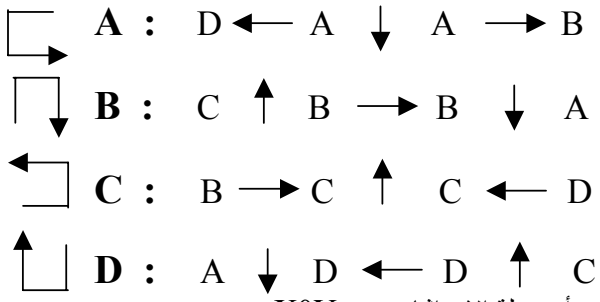
(منحنيات هيلبرت من H1 إلى H5)

نجد بعد دراسة الشكل السابق جيداً أنه مؤلف من خمسة أشكال متوضعة فوق بعضها , و يتألف كل من الثلاث الأولى منها من خط منكسر من إحدى الأشكال الثلاثة الموضحة كالتالي , حيث نرمز لهذه الخطوط بالرموز H1 و H2 و H3 .(خطوط هيلبرت من المستويات H1 و H2 و H3)



نلاحظ أنه يمكن الحصول على المستوى H_{i+1} من تركيب أربعة أشكال من المستوى H_i التي تعد أصغر حجماً بمرتين , و متوافقة من حيث الشكل , و توجه هذه الأشكال الأربعة إلى جميع الاتجاهات المختلفة , و تتصل مع بعضها بثلاثة قطع مستقيمة . يمكن اعتبار أن المستوى H_1 مؤلف من أربعة أشكال H_0 فارغة = أربعة نقاط تم وصلها بواسطة ثلاثة خطوط .

بما أن كل منحني H_i يتألف من أربعة منحنيات H_{i-1} متصلة , فإننا نستطيع التعبير عن إجرائية رسم H_i كتركيب لأربعة إجرائيات تقوم كل منها برسم H_{i-1} بالاتجاه و القياس المناسبين. لنرمز إلى هذه الإجرائيات الأربعة بـ A,B,C,D و لنرمز بأسهم إلى عمليات رسم الوصلات, و بالتالي نستطيع التعبير عن المخطط العودي لهذه الإجرائيات كما يلي:



لنكتب البرنامج حيث أن h طول القطعة المستخدمة في رسم الخط المنكسر و أن جملة الاحداثيات هي X0Y .

الحل: (حيث يمكن تطبيق البرنامج في Borland Turbo C++ under Dos)

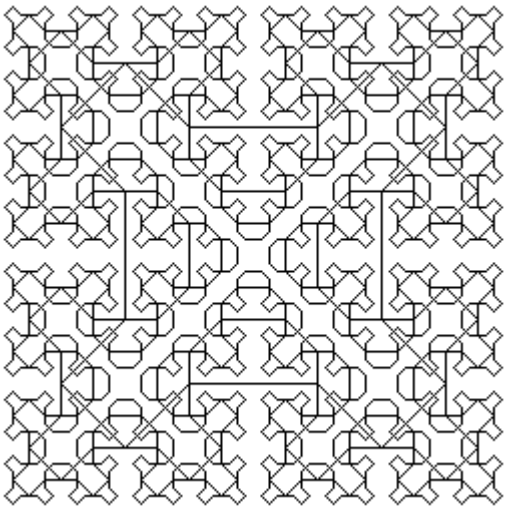
ملاحظات:

- التابع lineto(x,y) يرسم خط مستقيم بدءاً من الموقع الحالي لمؤشر الكتابة و حتى النقطة (x,y) .
- initgraph من أجل تهيئة الشاشة لإظهار الرسم البياني.
- setcolor لتغيير لون خط الرسم عندما تكون الشاشة في وضع Graphmode بينما Textcolor تستخدم في الوضع Textmode .

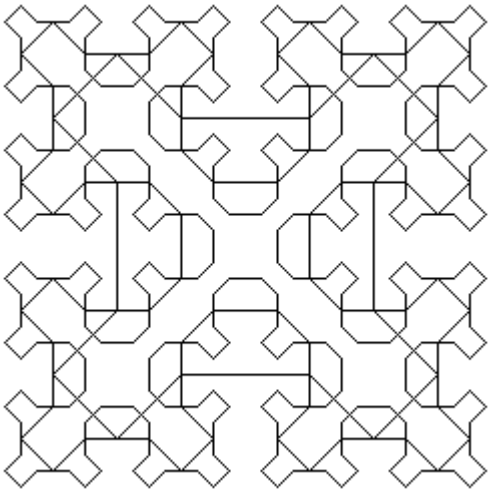
```
#include <graphics.h>
#include <conio.h>      //  --> getch()

const int n=5,h0=512;
int h,x,y,x0,y0;

void A(int i);
void B(int i);
void C(int i);
void D(int i);
/*************************************/
void main()
{
  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "");
  int i=0;
  h=h0;
  x0=h/2;
  y0=x0;
  do {
       i++;
       h/=2;
       setcolor(i);
       x0=x0+(h/2);
       y0=y0-(h/2);
       moveto(x0,y0);
       x=x0;
       y=y0;
       A(i); getch();
    }  while (i<n);
  closegraph();
}
/*************************************/
void A(int i)
{
  if (i>0)
     {
       D(i-1);  x-=h;  lineto(x,y);
       A(i-1);  y+=h;  lineto(x,y);
       A(i-1);  x+=h;  lineto(x,y);
       B(i-1);
     }
}
/*************************************/
void B(int i)
{
  if (i>0)
     {
       C(i-1);  y-=h;  lineto(x,y);
       B(i-1);  x+=h;  lineto(x,y);
       B(i-1);  y+=h;  lineto(x,y);
       A(i-1);
     }
}
/*************************************/
void C(int i)
{
  if (i>0)
     {
       B(i-1);  x+=h;  lineto(x,y);
       C(i-1);  y-=h;  lineto(x,y);
       C(i-1);  x-=h;  lineto(x,y);
       D(i-1);
     }
}
/*************************************/
void D(int i)
{
  if (i>0)
     {
       A(i-1);  y+=h;   lineto(x,y);
       D(i-1);  x-=h;   lineto(x,y);
       D(i-1);  y-=h;   lineto(x,y);
       C(i-1);
     }
}
```

❖ منحنيات سيربنسكي - Sierpinsky :

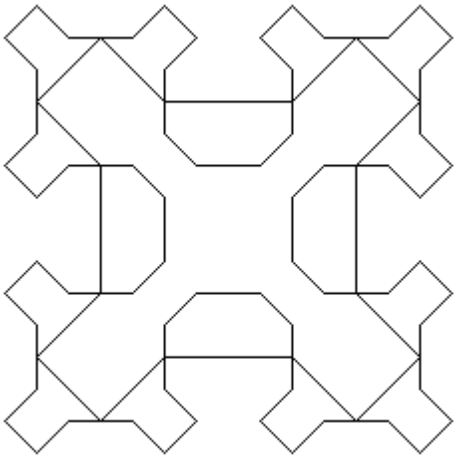
منحنيات سيربنسكي تشبه من حيث التركيب منحنيات هلبرت , لكنها أكثر تعقيداً و أناقة ! فمثلاً لدينا :



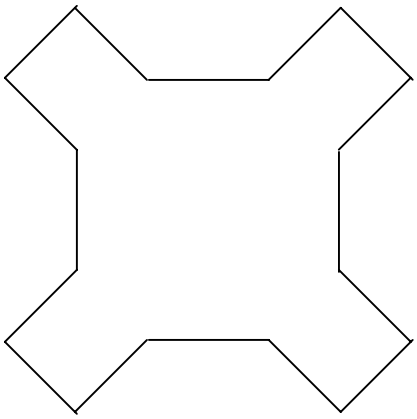
منحنيات سيربنسكي من S1 - S4



منحنيات سيربنسكي من S1 - S3



منحنيا سيربنسكي من S1 - S2

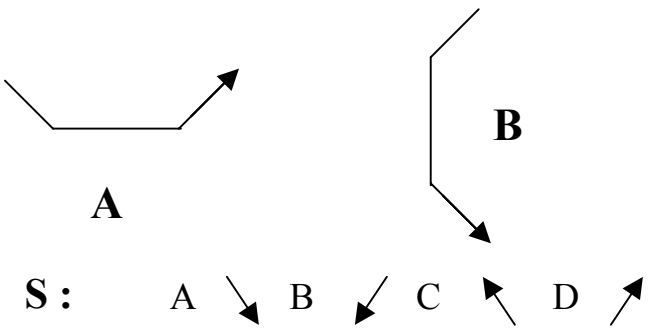


منحني سيربنسكي من المستوى S1

نلاحظ أنه لا يمكن رسم S2 بالاعتماد على S1 كون منحنيات سيربنسكي منحنيات مغلقة , لذلك يجب اعتماد مخطط عودي لخوارزمية الرسم , بحيث نحصل على منحنيات مفتوحة يؤدي وصلها إلى الحصول على الرسم المطلوب.

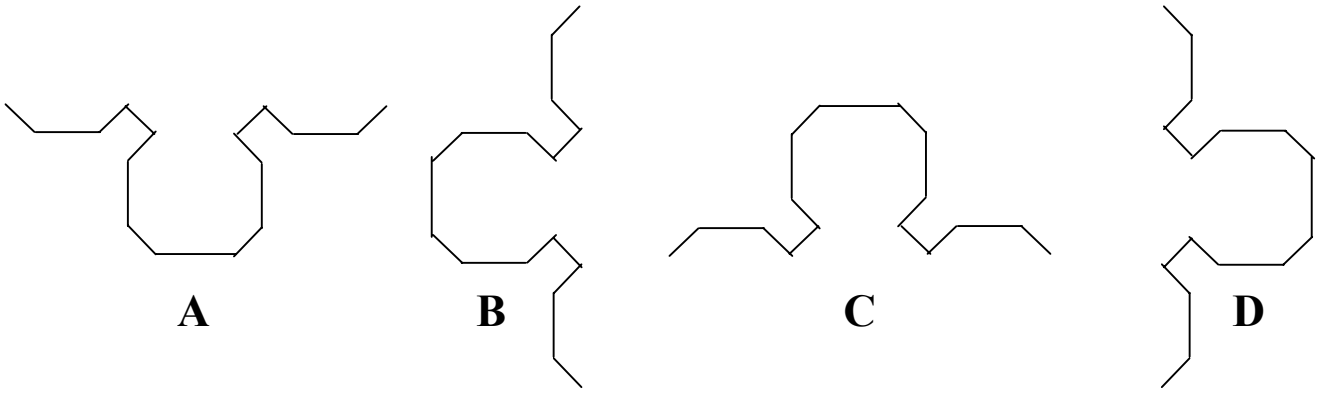
سنعتبر أن منحني سيربنسكي من الدرجة n هو تركيب لأربع إجرائيات A,B,C,D تجمعها أربع قطع مستقيمة لا تنتمي إلى هذه المنحنيات و يعطي المنحني المطلوب. إن المنحنيات الأربعة متماثلة و ينتج كل منها من الآخر بدوران قدره 90^o .

فمثلاً أثناء رسم S1 اعتمدنا على القطع الأربعة التالية مع الوصل بينها بقطع مستقيمة مائلة و ذلك بدئاً من عند رسم A :

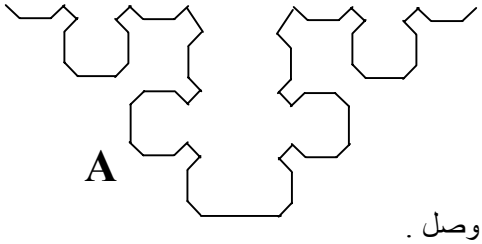


فالمخطط الأساسي لرسم منحني سيربنسكي (مهما كان المستوى) هو :

أما لرسم S2 يجب الاعتماد على القطع الأربعة التالية مع الوصل بينها بقطع مستقيمة مائلة و ذلك بدئاً من عند رسم A من المستوى الثاني:

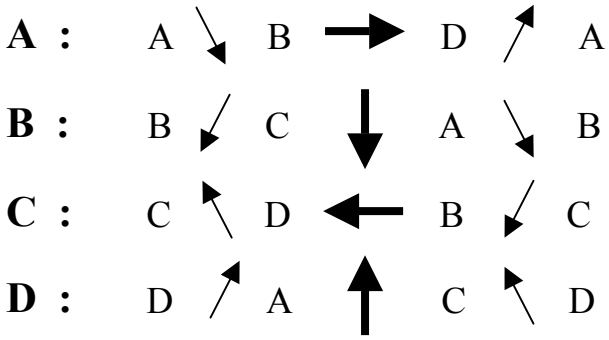


أما لرسم S3 يجب الاعتماد على القطع الأربعة التالية مع الوصل بينها بقطع مستقيمة مائلة و ذلك بدئاً من عند رسم A من المستوى الثالث:



. . . . . .

حيث أن منحني سيربنسكي من الدرجة صفر أي S0 هو عبارة عن أربعة نقاط تم الوصل بينها بقطع وصل .
أما المخطط العودي للإجرائيات الأربعة A,B,C,D التي تقوم برسم المنحنيات الجزئية فهو :



حيث يرمز السهم العريض ⟵ إلى قطعة مستقيمة مضاعفة الطول .
بإتباع نفس الخطوات التي أوردناها لرسم منحنيات هيلبرت نكتب البرنامج اللازم لرسم منحنيات سيربنسكي :
الحل: (حيث يمكن تطبيق البرنامج في Borland Turbo C++ under Dos)

```
#include <graphics.h>
#include <conio.h>

const int n=4,h0=256;
int i,h,x,y,x0,y0;

void A(int i);
void B(int i);
void C(int i);
void D(int i);

/*****************************************************************************/
void main()
{
  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "");
```

```
  i=0;
  h=h0/4;
  x0=2*h;
  y0=h;
  do {
       i++;
       x0-=h;
       h/=2;
       y0-=h;
       x=x0;
       y=y0;
       moveto(x0,y0);
       setcolor(i);
       A(i); x+=h; y+=h; lineto(x,y);
       B(i); x-=h; y+=h; lineto(x,y);
       C(i); x-=h; y-=h; lineto(x,y);
       D(i); x+=h; y-=h; lineto(x,y);
       getch();
     }  while (i<n);
  closegraph();
}
/*************************************************************************************/
void A(int i)
{
  if (i>0)
     {
       A(i-1);  x+=h;    y+=h; lineto(x,y);
       B(i-1);  x+=2*h;        lineto(x,y);
       D(i-1);  x+=h;    y-=h; lineto(x,y);
       A(i-1);
     }
}
/*************************************************************************************/
void B(int i)
{
  if (i>0)
     {
       B(i-1);  x-=h;  y+=h;    lineto(x,y);
       C(i-1);         y+=2*h;  lineto(x,y);
       A(i-1);  x+=h;  y+=h;    lineto(x,y);
       B(i-1);
     }
}
/*************************************************************************************/
void C(int i)
{
  if (i>0)
     {
       C(i-1);  x-=h;    y-=h;  lineto(x,y);
       D(i-1);  x-=2*h;         lineto(x,y);
       B(i-1);  x-=h;    y+=h;  lineto(x,y);
       C(i-1);
     }
}
/*************************************************************************************/
void D(int i)
{
  if (i>0)
```

```
    {
      D(i-1);  x+=h;  y-=h;    lineto(x,y);
      A(i-1);         y-=2*h;  lineto(x,y);
      C(i-1);  x-=h;  y-=h;    lineto(x,y);
      D(i-1);
    }
}
```

تمرين:

حساب معين مصفوفة مربعة عودياً باستخدام طريقة النشر و حساب الـ Minors , حيث أن الـ minor لعنصر x من مصفوفة مربعة a(n x n) هو المصفوفة الجزئية الناتجة من a بعد حذف كل من السطر و العمود اللذين يحتويان x (أي نحصل على مصفوفة ذات أبعاد n-1 x n-1).
نعرف معين (Determinant) لمصفوفة a مربعة من الدرجة n عودياً كما يلي:

إذا كانت a(1x1) أحادية فإن : det(a) = a[i,j] و إلا :

$$\det(a) = \sum_{i=0}^{n-1} (-1)^{i+j} \times a[i,j] \times \det\ (\text{minor}(a[i,j]))$$

حيث j عمود تم تثبيته .

الحل:

```
#include <iostream.h>

const int m=5;
struct matrix{          //  definition of quadratic matrix
       int n;          //  n= number of lines ( = number of columns )
       int val[m][m];
  };
matrix a;
int i,j;

int det(matrix a);
matrix minor(matrix a,int i,int j);
/*****************************************************************************/
void main()
{
  cout<<"Enter number of lines(=columns)= ";
  cin>>a.n;
  for (i=0;i<a.n;i++)
       for (j=0;j<a.n;j++)
           {
             cout<<"a["<<i<<"]["<<j<<"]=";
             cin>>a.val[i][j];
           }
  cout<<"Det(a)= "<<det(a);
}
/*****************************************************************************/
int det(matrix a)
{
  if (a.n==1) return a.val[0][0];
  else
      {
        matrix mat;
        int i,j=0,d=0;     //  j= the fixed column , d= det of a
        for (i=0;i<a.n;i++)
            {
              mat=minor(a,i,j);
              if (((i+j)%2)!=0)     //  (i+j) odd number
                  d-=a.val[i][j]*det(mat);
              else
```

```
                d+=a.val[i][j]*det(mat);
            }
        return d;
      }
}
/*****************************************************************************************/
matrix minor(matrix a,int i,int j)
{
  int ii,jj,t=-1,k;  //  t=line of new matrix  ,  k=column of new matrix
  matrix mat;
  mat.n=a.n-1;
  for (ii=0;ii<a.n;ii++)

      if (ii!=i)
         {
          t++;
          k=-1;
          for (jj=0;jj<a.n;jj++)
              if (jj!=j)
                 {
                   k++;
                   mat.val[t][k]=a.val[ii][jj];
                 }
         }
  return mat;
}
```

تحويل الخوارزميات العودية إلى خوارزميات تكرارية

1- طريقة حذف الاستدعاء العودي المتطرف:

نقول عن استدعاء عودي أنه متطرفاً إذا كان هذا الاستدعاء ينهي تنفيذ الإجرائية, أي لا يوجد أي تعليمة أخرى بعد تنفيذ الاستدعاء ضمن نص الإجرائية (بغط النظر عن موضع الاستدعاء العودي ضمن الإجرائية).

في حالة كون الاستدعاء العودي متطرفاً يمكن تحويله إلى خوارزمية تكرارية بالاستعانة بتعليمة goto أو حلقة تكرارية مثل while فيصبح الشكل العام للتحويل كالمخطط التالي :

```
void p( X … , Y … )
{          .
           .
     if (condition)  Y=…
     else
     p( E , Y );
           .
           .
}                استدعاء متطرف
```

```
void p( X … , Y … )
{   start:  .
            .
            .
        X = E ;
        goto start;
            .
            .
}
```

حيث X قائمة من متحولات الدخل , Y قائمة من متحولات الخرج , و الاستدعاء العودي p(E , Y) متطرف .

2- الطريقة العامة لتحويل الإجرائيات العودية إلى إجرائيات تكرارية:

سوف نؤجل هذه الطريقة حالياً إلى أن نصل بدراستنا إلى مفهوم المكدس Stack , لأنها تعتمد على استخدام المكدسات .

أمثلة على طريقة حذف الاستدعاء العودي المتطرف:

مثال (1) : حساب العاملي لعدد صحيح :

الخوارزمية العودية	استخدام goto label	استخدام حلقة while

long fact(int n)
{
  if ((n==1)||(n==0)) return 1;
  else
  return n*fact(n-1);
}
استدعاء متطرف
	long fact(int n)
	{ long f=1;
	Start:
	  if ((n==1)||(n==0))
	  return f;
	  else
	  { f*=n;
	  n--;
	  goto Start;
	  }
	}
		Long fact(int n)
		{
		  long f=1;
		  while (n!=0)
		  {
		  f*=n;
		  n--;
		  }
		  return f;
		}

مثال (2) : إيجاد القاسم المشترك الأكبر لعددين صحيحين:

الخوارزمية العودية	استخدام goto label	استخدام حلقة while

int gcd(int x,int y)
{
  if (y==0) return x;
  else
  return gcd(y,x%y);
}
استدعاء متطرف
	int gcd(int x,int y)
	{ int h;
	Start:
	  if (y==0) return x;
	  else
	  { h=x; x=y;
	  y=h%y;
	  goto Start;
	  }
	}
		int gcd(int x,int y)
		{ int h;
		  while (y!=0)
		  { h=x;
		  x=y;
		  y=h%y;
		  }
		  return x;
		}

مثال (3) : حساب رفع قوة لعدد صحيح:

الخوارزمية العودية	استخدام goto label	استخدام حلقة while

int power(int x,int n)
{
  if (n==0) return 1;
  else
  if (n==1) return x;
  else
  return x*power1(x,n-1);
}
استدعاء متطرف
	int power(int x,int n)
	{ int p=1;
	Start:
	  if (n==0) return p;
	  else
	  { p*=x;
	  n--;
	  goto Start;
	  }
	}
		int power(int x,int n)
		{ int p=1;
		  while (n!=0)
		  {
		  p*=x;
		  n--;
		  }
		  return p;
		}

مثال (4) : مسألة أبراج هانوي :

الخوارزمية العودية

```
void Hanoi(int n,char a,char b,char c)
{
  if (n==1) cout<<a<<"  -->  "<<c<<endl;
  else
     { Hanoi(n-1,a,c,b);
       cout<<a<<"  -->  "<<c<<endl;
       Hanoi(n-1,b,a,c);
     }
}
```

استدعاء متطرف

goto label

```
void Hanoi(int n,char a,char b,char c)
{ char h;
Start:
  if (n==1) cout<<a<<"  -->  "<<c<<endl;
  else
     { Hanoi(n-1,a,c,b);
       cout<<a<<"  -->  "<<c<<endl;
       n--;
       h=a; a=b; b=h;
       goto Start;
     }
}
```

While

```
void Hanoi(int n,char a,char b,char c)
{ char h;
  while (n!=1)
    { Hanoi(n-1,a,c,b);
      cout<<a<<"  -->  "<<c<<endl;
      n--;
      h=a; a=b; b=h;
    }
  cout<<a<<"  -->  "<<c<<endl;
}
```

مثال (5) : إيجاد جداء عددين صحيحين:

الخوارزمية العودية	استخدام goto label	استخدام حلقة while

`int multi(int x,int y)`
`{`
`  if (y==0) return 0;`
`  if (y==1) return x;`
`  else`
`  return x + multi(x,y-1);`
`}`
استدعاء متطرف
	`int multi(int x,int y)`
	`{  int m=x;`
	`  if (y==0) return 0;`
	`  else`
	`Start:`
	`  if (y==1) return m;`
	`  else`
	`  { m+=x;`
	`  y--;`
	`  goto Start; }}`
		`int multi(int x,int y)`
		`{ int m=x;`
		`  while (y!=1)`
		`  {`
		`  m+=x;`
		`  y--;`
		`  }`
		`  return m;`
		`}`

مثال (6) : خوارزمية البحث الثنائي (بطريقة تنصيف المجالات) عن عنصر x ضمن قائمة مرتبة t :

الخوارزمية العودية

```
char binS(int x,array t,int low,int high)
{ int mid;
  c='n';    //  no , x not found
  if (low<=high)
     {
       mid=(low+high)/2;
       if (x==t[mid]) c='y';   //  yes , x found
       else
       if (x<t[mid]) return binS(x,t,low,mid-1);
       else          return binS(x,t,mid+1,high);
     }
  return c;
}
```

goto label

```
char binS(int x,array t,int low,int high)
{  int mid;
   c='n';    //  no , x not found
Start:
   if (low<=high)
      {
        mid=(low+high)/2;
        if (x==t[mid]) c='y';   //  yes , x found
        else
        if (x<t[mid])
           { high=mid-1;
             goto Start;
           }
        else
           { low=mid+1;
             goto Start;
           }
      }
   return c;
}
```

While

```
char binS(int x,array t,int low,int high)
{
  int mid;
  char c='n';    //  no , x not found
  while ((low<=high)&&(c=='n'))
    {
      mid=(low+high)/2;
      if (x==t[mid]) c='y';   //  yes , x found
      else
      if (x<t[mid]) high=mid-1;
      else          low=mid+1;
    }
  return c;
}
```

الخوارزميات التراجعية - Backtracking Algorithms

الخوارزميات التراجعية هي خوارزميات عودية تعتمد طريقة "التجريب و الخطأ" (Try-and-Error) في حل المسائل .
تتلخص هذه الطريقة ببناء الحل النهائي للمسألة عن طريق مجموعة من الخطوات , في كل خطوة نحدد الإمكانيات المتاحة للخطوة التالية , ثم ندرس هذه الإمكانيات بأن ننتقي أحدها , و نسجلها على أنها الخطوة التالية في الحل النهائي , و نتابع الخوارزمية اعتماداً على هذه الخطوة . عندما يتأكد لنا أن اختيارنا لا يقود إلى الحل النهائي , أو يؤدي إلى طريق مسدود , نعدل عن هذه الخطوة . تشبه هذه العملية بناء سجل أخطاء يفيد في تجنب الوقوع في الخطأ مرتين .
إن المخطط العام للخوارزميات التراجعية كالتالي (حيث كتب بلغة الخوارزميات pseudocode , أي دون تحديد لغة برمجية معينة , بل تم وضع الشكل العام للخوارزمية مع الشرح باللغة الإنكليزية و دون تفاصيل دقيقة) :

```
void  try()
{
 initialize selection of candidates;
 do {
      select next candidate;
      if (acceptable)
         {
           record it;
           if (solution incomplete)
              {
                try next step;
                if (not successful)
                    cancel recording;
              }
         }
 } while (not successful)and(remains of
                                candidates);
}
```

```
void  try()
{
   تهيئة الإمكانيات الممكن ترشيحها ;
  do {
       اختيار المرشح التالي ;
      if (مقبول)
         {
            تسجيل الخطوة الحالية ;
           if (الحل غير مكتمل)
              {
                 حاول الخطوة التالية ;
                if (غير ناجحة)
                    إلغاء تسجيل آخر خطوة;
              }
         }
  }while (الحل غير مكتمل)and(وجود مرشحين أخرى);
}
```

تختلف تفصيلات هذا المخطط باختلاف المسألة المطروحة . سنتداول هذا المخطط في الأمثلة التالية لنبين استخداماته المختلفة من خلال عرض حلول بعض أهم المسائل الشهيرة .

مسألة جولة حصان الشطرنج - Knightstour :

لدينا رقعة شطرنج فيها n×n مربعاً , نضع حصاناً في موقع معين $<X_0,Y_0>$ حيث : $0 \le X_0 \le n-1$ و $0 \le Y_0 \le n-1$ و علينا إيجاد طريقة لتغطية الرقعة (إذا كان ذلك ممكناً) أي أن نوجد متتالية من الحركات عددها n^2 - 1 بحيث يحدث المرور بكل مربع مرة واحدة فقط.
لحل المسألة سنهتم فقط بحركة الحصان التالية : سنفترض في كل مرحلة أننا قمنا بعدد من الحركات , و لدينا مجموعة من الإمكانيات التي يجب أن نجربها . في كل مرحلة لدينا عدد من الحركات الممكنة , نجرب إحدى هذه الحركات و نناقش "أتؤدي إلى حل أم إلى طريق مسدود؟" .
يمكن التعبير عن ذلك بالخوارزمية العودية المبسطة التالية و المستوحاة من المخطط الخوارزمي التراجعي العام:

```
void  try next move()
{
  initialize selection of moves;
  do {
       select next candidate from list of next moves;
       if (acceptable)
          {
            record move;
            if (board not full)
               { try next move;
                 if (not successful)
                    erase previous recording;
               }
          }
   } while (board not full)and(there are more candidates yet);
}
```

لتفصيل الخوارزمية سنحدد بنى المعطيات المستخدمة و معاملات الإجرائية . نمثل الرقعة بمصفوفة مربعة h نعرفها بالشكل :

```
const int n=5;
int   h[n][n];
```

حيث نعتبر : - h[X,Y] = 0 عندما لا يحدث المرور في المربع (X,Y)
- h[X,Y] = i عندما يحدث المرور في المربع (X,Y) في الخطوة i

معاملات الخوارزمية :

- الموقع الذي نريد تطبيق الخوارزمية فيه : (X,Y)
- رقم الخطوة : i
- متحول خرج q يعطي نتيجة المناقشة : نجاح أو فشل

بناءً على ذلك يمكن تبسيط العبارة : "الرقعة ليست ممتلئة" (board not full) بالشكل : $i < n^2$

إذا استعملنا متحولين موضعيين u , v لتمثيل إحدى المواقع الممكنة (حسب طريقة حركة الحصان المعروفة على الشكل L) فإن العبارة "مقبول" (acceptable) يمكن التعبير عنها بالشكل : $0 \le v \le n-1$ and $0 \le u \le n-1$ and $h[u,v] = 0$.

المرحلة الأخيرة من تبسيط الخوارزمية هي تحديد الحركات الممكنة التي يمكن إجراؤها من موقع معين <X,Y> . نعلم من قواعد لعبة الشطرنج أن الحصان يستطيع في الحالة العامة الانتقال إلى ثمانية مواقع يبينها الشكل التالي :

		3		2			
	4				1		
			X				
	5				8		
		6		7			

يمكن الحصول على إحداثيات هذه المواقع الجديدة بإضافة فروق إلى إحداثيات موقع الحصان و نخزن هذه الفروق في جدولين a و b يعرفان بالشكل :

```
int  a[8] , b[8];
```

كما نستطيع استخدام عداد k لترقيم إمكانيات الانتقال التالي . عند استدعاء الإجرائية نحدد المعاملين <X,Y> اللذين يمثلان الموقع الابتدائي للحصان , و نسند القيمة واحد للموقع h[X,Y] .

البرنامج التالي يعطي الحل الكامل لمسألة جولة الحصان على رقعة الشطرنج , حيث يبدأ الحصان من الموقع h[0,0] .

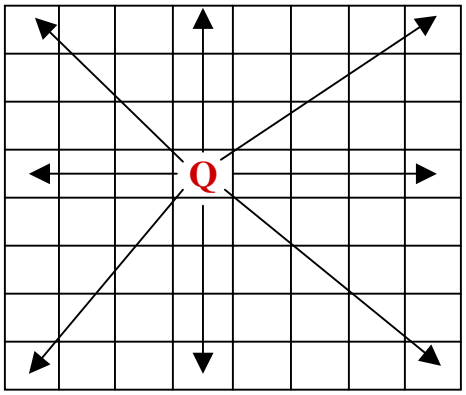
```
#include <iostream.h>
#include <iomanip.h>   // --> setw()
const int n=5;      //  number of lines/columns
int  i,j;
char q='n';       // no , the solution is incomplete
int  a[8],b[8];
int  h[n][n]={0};
void try1(int i,int x,int y,char &q);
/*************************************************************************************/
void main()
{ a[0]=2;   b[0]=1;
  a[1]=1;   b[1]=2;
  a[2]=-1;  b[2]=2;
  a[3]=-2;  b[3]=1;
  a[4]=-2;  b[4]=-1;
  a[5]=-1;  b[5]=-2;
  a[6]=1;   b[6]=-2;
  a[7]=2;   b[7]=-1;
  h[0][0]=1;         // first step (i=1) is fixed
```

```
  try1(2,0,0,q);
  if (q=='y')
     for (i=0;i<n;i++)
         {
           for (j=0;j<n;j++) cout<<setw(5)<<h[i][j];

           cout<<endl;
         }
  else     cout<<"No solution !\n";
}
/*************************************************************************************************/
void try1(int i,int x,int y,char &q)
{
  int u,v,k=-1;
  do {
       k++;
       u=x+a[k];  v=y+b[k];
       if ((u>=0)&&(u<n)&&(v>=0)&&(v<n)&&(h[u][v]==0))
          {
            h[u][v]=i;
            if (i<n*n)
               {
                 try1(i+1,u,v,q);
                 if (q=='n')
                    h[u][v]=0;
               }
            else
               q='y';           //  yes , the solution is complete
          }
       } while ((q=='n')&&(k<7));
}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

❑ مسألة الوزراء الثمانية - EightQueens :

الغرض من هذه المسألة هو توزيع ثمانية وزراء على رقعة شطرنج , بحيث لا يكون فيها أي واحد مهدداً للآخر . بحسب المخطط العام للخوارزميات التراجعية فإن شكل الإجرائية هو :

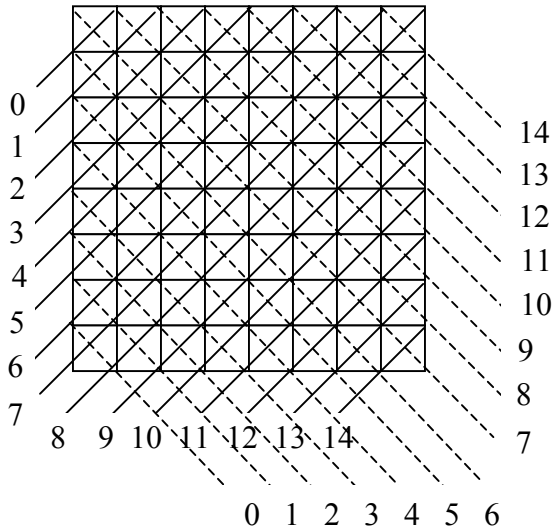
```
void  try(int i)
{
  initialize selection of positions for i-th queen;
  do {
       make next selection;
       if (safe)
          {
            setqueen;
            if (i<7)
               {
                 try (i+1);
                 if (not successful)
                    remove queen;
               }
          }
   } while (not successful)and(there are more positions yet);
}
```



نعلم من قواعد الشطرنج أنه يمكن للوزير أن يهدد جميع المربعات الموجودة في العامود أو السطر أو القطر نفسه , كما يبين ذلك الشكل المجاور . لذا يجب وضع وزير واحد في كل عامود , و تؤدي عملية تحديد موقع الوزير رقم i إلى تحديد رقم السطر الذي سيوضع فيه ضمن العامود رقم i .
لحل هذه المسألة على الرقعة لا بد من مناقشة عميقة لبنية المعطيات المستخدمة , بحيث تُحدد بنية تساعد في إجراء الاختبارات اللازمة عند تقرير وضع الوزير في مربع معين . في البداية يمكن تعريف رقعة الشطرنج بأنها مصفوفة مربعة , لكن هذا التمثيل سيؤدي إلى اختبارات صعبة لنعرف أيكون مربع ما آمناً (غير مهدداً) أم لا .
إن ما يهمنا هو موقع كل وزير ضمن العامود الموافق , و أن نعرف أيحوي سطر معين أو قطر معين وزيراً أم لا . لذا سنستخدم بنى المعطيات التالية :

```
int   x[8];       //    i  تحوي رقم السطر الذي يوضع فيه الوزير رقم  x[i]
char  a[8];       //                تعني أن السطر j لا يحوي أي وزير  a[j]=='y'
char  b[15];      //    تعني أن القطر المباشر رقم j لا يحوي أي وزير  b[j]=='y'
char  c[15];      //    تعني أن القطر المتعامد رقم j لا يحوي أي وزير  c[j]=='y'
```

يبين الشكل التالي طريقة ترقيم أقطار رقعة الشطرنج :



بعد تحديد بنى المعطيات المستخدمة نستطيع تفصيل الخوارزمية , فنلاحظ أن المواقع الممكنة للوزير رقم i هي كل مربعات العامود رقم i (عددها ثمانية).
لاختبار هذه المواقع نستعمل عداداً j يكون في البداية صفراً و يزداد في كل مرة واحداً حتى يصل إلى القيمة 7 .

تكافئ عملية وضع الوزير رقم i في السطر j تنفيذ التعليمات التالية :

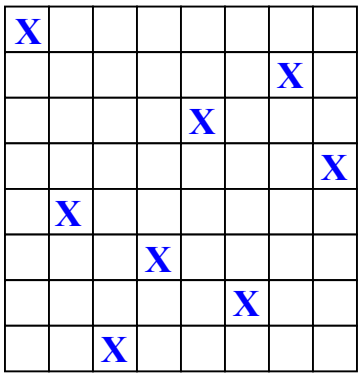
```
x[i]=j;
a[j]='n';
b[7+i-j]='n';
c[i+j]='n';
```

و تكافئ عملية رفع الوزير رقم i من السطر j تنفيذ التعليمات التالية :

```
a[j]='y';
b[7+i-j]='y';
c[i+j]='y';
```

و يمكن التعبير عن القضية "أمان" (safe) بالشكل : `((a[j]=='y')&&(b[7+i-j]=='y')&&(c[i+j]=='y'))`

يعطي البرنامج التالي الحل للمسألة الممثل بالشكل : x = [0 , 4 , 7 , 5 , 2 , 6 , 1 , 3]



إحدى حلول مسألة الوزراء الثمانية

الحل:

```
#include <iostream.h>
#include <iomanip.h>

int  i;
char q='n';       //  no , there is no solution
char a[8];       //  lines
char b[15];     //  direct diagonals
char c[15];    //  indirect diagonals
int  x[8];    //  solution = lines

void try1(int i,char &q);
void print();
/*************************************************************************************/

void main()
{ for (i=0;i<8;i++)  a[i]='y';      //  all lines are safe
  for (i=0;i<15;i++) b[i]='y';     //  all direct diagonals are safe
  for (i=0;i<15;i++) c[i]='y';    //  all indirect diagonals are safe
  try1(0,q);
  if (q=='y') print();
  else        cout<<"No solution !\n";
}
/*************************************************************************************/
void try1(int i,char &q)
{
  int j=-1;
  do {
       j++;
       q='n';
       if ((a[j]=='y')&&(b[7+i-j]=='y')&&(c[i+j]=='y'))
          {
            x[i]=j;
            a[j]='n';
            b[7+i-j]='n';
            c[i+j]='n';
            if (i<7)
               {
                 try1(i+1,q);
                 if (q=='n')
                    {
                      a[j]='y';
                      b[7+i-j]='y';
                      c[i+j]='y';
                    }
               }
            else
               q='y';
          }
   } while ((q=='n')&&(j<7));
}
/*************************************************************************************/
void print()
{ for (int k=0;k<8;k++)
      cout<<setw(5)<<x[k];
  cout<<endl;
}
```

نستطيع الاستفادة من خوارزمية حل مسألة الوزراء الثمانية في تعميم المخطط العام للخوارزميات التراجعية , بحيث نوجد جميع الحلول لمسألة معينة. و لتحقيق ذلك يجب عدم إيقاف تجريب الإمكانيات المتاحة عند الوصول إلى حل, و إنما نسجل هذا الحل و نتابع تجريب الإمكانيات الأخرى. يصبح مخطط الخوارزميات التراجعية التي توجد جميع الحلول كالتالي :

```
void try(int i)
{
  for (int k=0;k<m-1;k++)
      {
        select k-th candidate;
        if (acceptable)
           {
             record it;
             if (i<n)
                try(i+1);

             else
                print solution;
           }
      }
}
```

في حالة مسألة الوزراء الثمانية تصبح إجرائية البحث عن الحلول كالتالي :

```
void try2(int i)
{
  for (int j=0;j<8;j++)
      {
        if ((a[j]=='y')&&(b[7+i-j]=='y')&&(c[i+j]=='y'))
           {
             x[i]=j;
             a[j]='n';
             b[7+i-j]='n';
             c[i+j]='n';
             if (i<7) try2(i+1);
             else     print();
             a[j]='y';
             b[7+i-j]='y';
             c[i+j]='y';
           }
      }
}
```

بهذا نستطيع إيجاد كافة الحلول و عددها 92 حل , من بين هذه الحلول يوجد 12 حلاً مستقلاً (لا ينتج أحدها عن الآخر بتدوير الرقعة أو بالتناظر) .

~~~~~~
~~~~~~

▫ مسألة الخيار الأمثل - Optimal Selection :

يأتي هذا المثال ليبين كيف يمكن أن نستخدم المخطط العام للخوارزميات التراجعية في مقارنة الحلول التي يمكن إيجادها لمسألة معينة. درسنا في المثالين السابقين كيف يمكن استخدام هذا المخطط لإيجاد حل واحد, كما فعلنا في مسألة جولة حصان الشطرنج, أو في مسألة الوزراء الثمانية, ثم عممنا هذا المخطط لنستطيع إيجاد جميع الحلول لمسألة الوزراء الثمانية. في هذه الفقرة سنتبع الطريقة نفسها , لكننا سنحتفظ بحل واحد يكون أمثلياً اعتماداً على معايير تسمح بمقارنة الحلول الممكنة .
لإيجاد الحل الأمثل لمسألة معينة يجب إيجاد جميع الحلول الممكنة بحيث نحتفظ دوماً بحل واحد يُعتبر أمثلياً بحسب معيار معين. إذا افترضنا أننا نستطيع مقارنة الحلول بواسطة تابع موجب F(S) حيث (S إحدى الحلول) , عندئذ يمكن الحصول على الحل الأمثل بتعديل المخطط العام للخوارزميات التراجعية التي توجد جميع الحلول بحيث نستبدل عملية إظهار الحل بالتعليمات التالية :

```
if ( F(Solution)>F(Optimal) )
   Optimal=Solution;
```

و بذلك نحتفظ في كل لحظة بأفضل حل أمكن الحصول عليه في المتحول Optimal .

سنختار مثالاً لمسألة الحل الأمثل مسألة هامة تتلخص بإيجاد الاختيار الأمثل من مجموعة عناصر معطاة. تجري عملية بناء الخيارات الممثلة للحلول المقبولة بالتدريج , عن طريق مناقشة كل عنصر من عناصر المجموعة الأساسية. نناقش في حالة كل عنصر فائدة ضم هذا العنصر إلى الاختيار أو عدم فائدته, ثم ننتقل إلى العنصر التالي عودياً.

عند فحص فائدة عنصر معين نستنتج أحد أمرين : إما أن نضم هذا العنصر إلى الخيار الآني الذي نحن بصدد بنائه, أو نستبعد هذا العنصر و نتابع بناء اختيارنا دونه. و يجب مناقشة هذين الاحتمالين كل على حدة , و هذا مما يجعل خوارزمية الحل تأخذ الشكل التالي :

```
void try(int i)
{
  if (inclusion is acceptable)
     {
       include i-th object;
       if  (i<n)
           try(i+1);
       else
           check optimality;
       erase i-th object;
     }
  if (exclusion is acceptable)    //        إذا كان الاستبعاد مقبولاً
     if  (i<n)
         try(i+1);
     else
         check optimality;
}
```

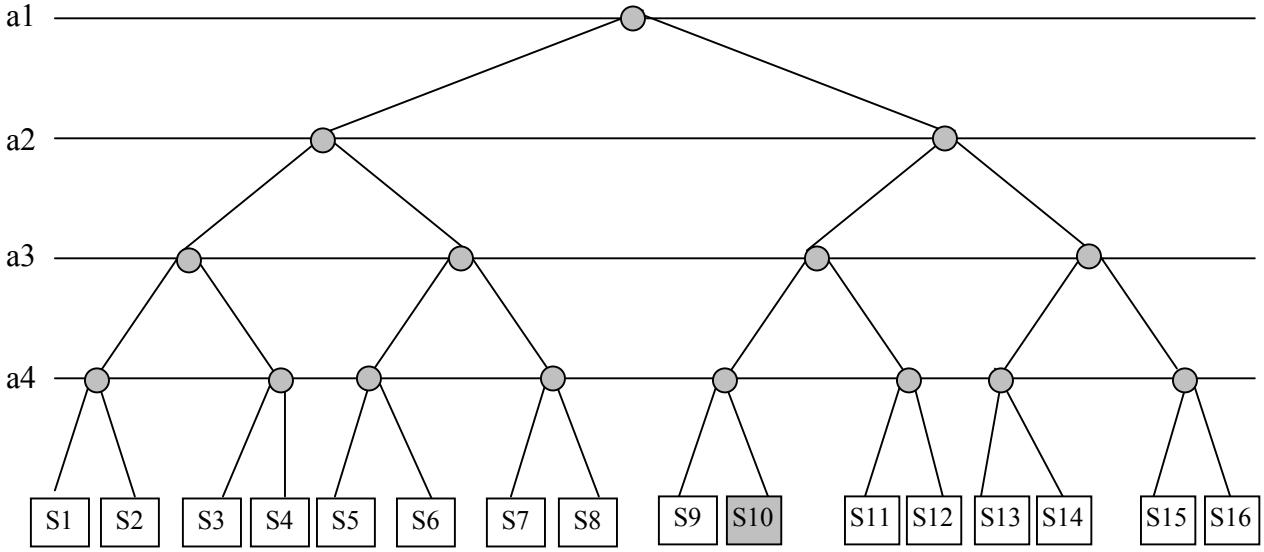
تبين دراسة هذه الخوارزمية أن هناك 2^n اختياراً ممكناً (عدد أجزاء مجموعة مؤلفة من n عنصر) لذلك يجب أن نستفيد من معايير القبول في الحد كثيراً من هذا العدد . سنوضح طريقة تقليص هذا العدد في ضوء مثال محدد .

لتكن A = { a1 , a2 , ... , an } مجموعة من الأشياء لكل منها وزن W و قيمة V , و لنفترض أننا نريد بناء مجموعة جزئية من A بحيث يكون مجموع أوزان عناصرها أقل من حد معين , و مجموع أسعارها أعظمياً. تصادف هذه المسألة أي مسافر يستعيد للسفر بالطائرة, و يقوم بإعداد حقيبته, و عليه اختيار مجموعة صغيرة من الأشياء لا يتجاوز وزنها العام الوزن المسموح به في المطار .
لتوضيح مبدأ العمل نأخذ حالة بسيطة, تحوي المجموعة فيها أربعة عناصر, يبين الجدول التالي وزن كل منها و قيمته :

العنصر	a1	a2	a3	a4
الوزن (W)	10	11	15	17
القيمة (V)	18	20	25	17

و نريد اختيار مجموعة جزئية من هذه العناصر بحيث لا يزيد وزنها الكلي عن 30 كغ و تكون قيمتها أعظمية. يمكن تمثيل تنفيذ خوارزمية البحث عن أفضل اختيار بالشجرة المبينة بالشكل التالي , تمثل كل عقدة داخلية في هذه الشجرة عملية مناقشة ضم أو استبعاد عنصر. (الفرع اليساري يعني

ضم العنصر و الفرع اليميني يعني استبعاد العنصر). و تمثل الأوراق الخيارات الناتجة أو المجموعات الجزئية للمجموعة A (عددها 16). من دراسة هذه الحلول يتبين أنه يجب استبعاد الحلول S1 , S2 , S3 , S5 , S9 , S13 لأنها لا تحقق شرط الوزن. أما بقية الحلول فيجب حساب قيمة كل منها, و من الواضح من أن الحل S10 هو أفضل حل , و يعطي القيمة 45 .



S1={a1,a2,a3,a4}; S2={a1,a2,a3}; S3={a1,a2,a4}; S4={a1,a2}; S5={a1,a3,a4}; S6={a1,a3};
S7={a1,a4}; S8={a1}; S9={a2,a3,a4}; S10={a2,a3}; S11={a2,a4}; S12={a2};
S13={a3,a4}; S14={a3}; S15={a4}; S16={ };

سوف نستخدم لحل المسألة المتحولات التالية :

limW : الوزن الأقصى لـ n عنصر (= ثابت)
totV : السعر الكلي لـ n عنصر (= ثابت)
s : صف يجمع عناصر الاختيار الآني
opts : صف يعبر عن الاختيار الأمثل الذي حصلنا عليه حتى اللحظة الحالية
maxV : مجموع أسعار عناصر الاختيار الأمثل opts
av : السعر العام للاختيار s الذي حصلنا عليه حتى اللحظة الحالية
tw : الوزن العام للاختيار s الذي حصلنا عليه حتى اللحظة الحالية

لنحدد طريقة لتقرير ضم عنصر إلى الصف s الذي نحن بصدد بنائه أو عدم ضمه :
لنمثل العناصر المنضمة بصف s مؤلف من أعداد صحيحة عددها نفس عدد العناصر الكلي , و نعطي لها قيم بدائية "0" . فكلما نريد ضم عنصر ترتيبه i نعطي للعنصر المقابل له في الصف s أي s[i] القيمة "1". أما في حال عدم ضم العنصر يبقى عنصر الصف المقابل له "0". ففي كل خطوة نمشيها سنناقش ضم عنصر واحد إلى الصف s أم لا (حيث أن العناصر مرقمة من 0 إلى n-1).

لنفترض tw الوزن الكلي للعناصر المنضمة في الصف s و أن av هو مجموع قيم العناصر المنضمة إضافة إلى قيم العناصر التي لم نناقش بعد ضمها إلى الصف . و بفرض i رقم العنصر الذي نعالجه حالياً نجد :
في البداية i = 0 :

$$tw = 0; \qquad av = \sum_{i=0}^{n-1} a[i].v$$

و عندما تصبح i = n-1 يكون :

$$tw \le limW; \qquad av = \sum_{s[i]=1} a[i].v$$

عندئذ يمكن التعبير عن الشرط "الضم مقبول" بالقضية : $tw + a[i].w \le limW$
و يمكن التعبير عن الشرط "الاستبعاد مقبول" بالقضية : $av - a[i].v > maxV$

و الذي يكافئ القول إن مجموع قيم العناصر الموجودة حالياً في s و التي يمكن ضمها مستقبلاً يمكن أن يتجاوز أفضل قيمة وصلنا إليها حتى الآن.
عندما تكون i = n-1 و يكون الاستبعاد مقبولاً , فإنه من المؤكد أن الحل الناتج أفضل من آخر حل أمكن الحصول عليه .

البرنامج التالي يعطي الحل الكامل لهذه المسألة حيث يبين الجدول التالي حالة انضمام 12 عنصر , حيث أعطينا الأوزان العظمى قيماً بين 10 و 100 كيلوغراماً .

```
#include <iostream.h>
#include <iomanip.h>   // --> setw()

const n=12;
struct objects {
       int v,w;    // v=value , w=weight
  };
int i;
objects a[n];
int limW,totV=0,maxV;
int w1,w2,w3;
int s[n]={0},opts[n]={0}; // if (s[i]==1) object i selected , if (s[i]==0) not selected

void try1(int i,int tw,int av);
/*****************************************************************************/
void main()
{ for (i=0;i<n;i++)
      { cout<<"Enter weight,value ("<<i<<") = ";
        cin>>a[i].w>>a[i].v;
        totV+=a[i].v;
      }
  cout<<"Enter w1,w2,w3 = ";
  cin>>w1>>w2>>w3;
  cout<<"Weight ";
  for (i=0;i<n;i++)
      cout<<setw(4)<<a[i].w;
  cout<<endl;

  cout<<"Value  ";
  for (i=0;i<n;i++)
      cout<<setw(4)<<a[i].v;
  cout<<endl;
  do {
       limW=w1;
       maxV=0;
       try1(0,0,totV);
       cout<<setw(6)<<limW;
       for (i=0;i<n;i++)
           if (opts[i]==1) cout<<setw(4)<<"*";
           else            cout<<setw(4)<<" ";
       cout<<endl;
       w1+=w2;
   } while (w1<=w3);
}
/*****************************************************************************/
void try1(int i,int tw,int av)
{
  int av1;
  if (tw+a[i].w<=limW)
     {
       s[i]=1;
       if (i<n-1) try1(i+1,tw+a[i].w,av);
```

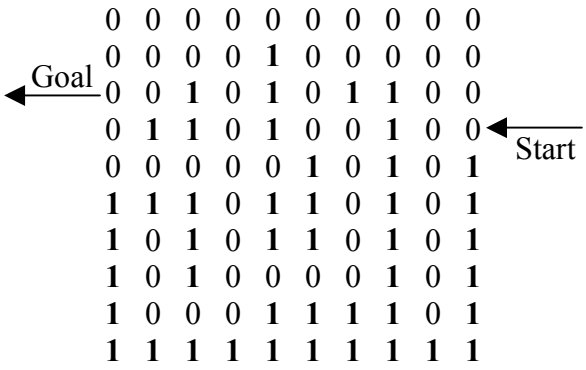
```
        else
        if (av>maxV)
           {
             maxV=av;
             for (int j=0;j<n;j++)
                 opts[j]=s[j];
           }
        s[i]=0;
      }
  av1=av-a[i].v;
  if (av1>maxV)
     if (i<n-1)
        try1(i+1,tw,av1);
     else
        {
          maxV=av1;
          for (int j=0;j<n;j++)
              opts[j]=s[j];
        }
}
```

يكون تنفيذ البرنامج كالجدول التالي :

21	20	19	18	17	16	15	14	13	12	11	10	الوزن
25	24	24	23	21	19	20	21	17	15	18	15	القيمة
											*	10
		*										20
		*								*		30
						*	*			*		40
						*	*			*	*	50
							*	*	*	*	*	60
		*				*	*			*	*	70
			*			*	*		*	*	*	80
*		*				*	*			*	*	90
		*	*			*	*	*		*	*	100

تمرين :

يمثل الشكل التالي متاهة , حيث أن الواحدات تمثل جدران المتاهة و الأصفار تمثل الممرات الممكن سلوكها عبر المتاهة . و بالتالي يمكن الانتقال ضمن المتاهة من موضع إلى موضع آخر يوجد فيه صفراً .



من أجل إيجاد المخرج من هذه المتاهة نضع يدنا اليمنى على الجدار الأيمن عند المدخل Start و نقوم بالسير قدماً دون أن نرفع يدنا فوق الأصفار , و نستمر بالمشي حتى أن نصل إلى المخرج Goal . و المطلوب:

اكتب برنامجاً يعتمد على الخوارزمية التراجعية try1 التي تقوم بإيجاد الطريق الواصل ما بين المدخل و المخرج , حيث يجب في نهاية البرنامج طباعة الشكل السابق نفسه لكن مكان الأصفار الواقعة على الطريق يجب وضع رقم الخطوة الحالية لمعرفة تسلسل الخطوات . (ثم عمم المسألة لإيجاد جميع الحلول و ثم الحل الأمثل فقط) .

الحل:

```
#include <iostream.h>
#include <conio.h>

const int n=10;     //  number of lines/columns

int h[n][n]={{0,0,0,0,0,0,0,0,0,0},{0,0,0,0,1,0,0,0,0,0},{0,0,1,0,1,0,1,1,0,0},
             {0,1,1,0,1,0,0,1,0,0},{0,0,0,0,0,1,0,1,0,1},{1,1,1,0,1,1,0,1,0,1},
             {1,0,1,0,1,1,0,1,0,1},{1,0,1,0,0,0,0,1,0,1},{1,0,0,0,1,1,1,1,0,1},
             {1,1,1,1,1,1,1,1,1,1}};
int optS[n][n];  // optimal solution
int a[4],b[4];
int i,j,Xn,Yn,minL=0; //  minL = length of optimal solution (=shortest way)
char q='n',c;         // no , the solution is incomplete

void try1(int i,int x,int y,char &q);  // one solution
void try2(int i,int x,int y);          // all solutions
void try3(int i,int x,int y);         // optimal solution
void print(int h[n][n]);
/*************************************************************************************/
void main()
{
  a[0]=-1; b[0]=0;
  a[1]=0;  b[1]=1;
  a[2]=1;  b[2]=0;
  a[3]=0;  b[3]=-1;
  cout<<"Enter Xo = "; cin>>i;
  cout<<"Enter yo = "; cin>>j;
  h[i][j]=2;
  cout<<"Enter X-Goal = "; cin>>Xn;
  cout<<"Enter y-Goal = "; cin>>Yn;
  cout<<"One Solution      -->  Press 1\n";
  cout<<"All Solutions     -->  Press 2\n";
  cout<<"Optimal Solution  -->  Press 3\n";
  c=getch();
  if (c=='1')
     {
       try1(3,i,j,q);
       if (q=='y')
           print(h);
       else
           cout<<"No solution";
```

```
     }
  else
  if (c=='2')
     try2(3,i,j);
  else

     {
       try3(3,i,j);
       print(optS);
     }
  getch();
}
/*****************************************************************************************/
void try1(int i,int x,int y,char &q)
{
  int u,v,k=-1;
  do{
      ++k;
      u=x+a[k];  v=y+b[k];
      if ((u>=0)&&(u<n)&&(v>=0)&&(v<n)&&(h[u][v]==0))
         {
           h[u][v]=i;
           if ((u==Xn)&&(v==Yn)) q='y';
           else
              {
                try1(i+1,u,v,q);
                if (q=='n') h[u][v]=0;
              }
         }
  } while ((q=='n')&&(k<3));
}
/*****************************************************************************************/
void try2(int i,int x,int y)
{
  int u,v,k=-1;
  do{
      ++k;
      u=x+a[k];  v=y+b[k];
      if ((u>=0)&&(u<n)&&(v>=0)&&(v<n)&&(h[u][v]==0))
         {
           h[u][v]=i;
           if ((u==Xn)&&(v==Yn))
              {
                print(h);
                getch();
              }
           else
                try2(i+1,u,v);
           h[u][v]=0;
         }
  } while (k<3);
}
/*****************************************************************************************/
void try3(int i,int x,int y)
{ int u,v,k=-1;
  do{
      ++k;
      u=x+a[k];  v=y+b[k];
```

```
      if ((u>=0)&&(u<n)&&(v>=0)&&(v<n)&&(h[u][v]==0))
         {
           h[u][v]=i;
           if ((u==Xn)&&(v==Yn))
              {
                if ((minL==0)||(i<minL))
                   {
                     minL=i;
                     for (int w=0;w<n;w++)
                         for (int t=0;t<n;t++)
                             optS[w][t]=h[w][t];

                   }
              }
           else
                try3(i+1,u,v);
           h[u][v]=0;
         }
  } while (k<3);
}
/*************************************************************************************/
void print(int h[n][n])
{
  for (int i=0;i<n;i++)
      {
        for (int j=0;j<n;j++)
            cout<<h[i][j]<<"    ";
        cout<<endl;
      }
}
```

❑ **تمرين للإطلاع عن الخوارزميات التراجعية:**

مسألة الزواج المستقر - The stable marriage problem :

لتكن لدينا المجموعتان غير المتقاطعتان A و B , و بعدد أساسي واحد و يساوي n . و المطلوب:
إيجاد مجموعة أزواج مرتبة من المجموعة n : 1 و من الشكل $< a , b >$ بحيث يكون $a \in A$ و $b \in B$, و يحققان بعض الشروط الأخرى. توجد معايير مختلفة لاختيار هذه الأزواج , و سوف نتعرف في الفقرة التالية على أحدها , و الذي يدعى قاعدة الزواج المستقر stable marriage rule .
لنفرض أن B مجموعة شباب , و A مجموعة آنسات , و لنفرض أن كل شخص من كلتا المجموعتين قد وضع ترتيباً محدداً بالمواصفات التي يرغب بأن تتحقق لدى قرينه الممكن الزواج منه . فإذا أمكن اختيار n زوجاً من الثنائيات $< a ,b >$ بحيث يكون الرجل a و الآنسة b غير متزوجين من بعضهما , و يرغب كل منهما في أن يكون قرين الآخر , عندئذ تعد مجموعة القران تلك غير مستقرة . و في حال عدم وجود هذه الأزواج تدعى المجموعة مستقرة .
تعد هذه المسألة حالة نوعية للكثير من المسائل المشابهة , والتي يتعلق فيها التوزيع ببعض الأفكار و الآراء , كعملية اختيار طلاب كلية (المفاضلة) , أو عملية فرز عناصر فرقة عسكرية , أو غيرها من المسائل المشابهة .
إن مثال اختيار الشريك الزوجي مثال ذو حدس شخصي , و لكن نلاحظ أن عملية وضع قائمة رغبات من كلا الطرفين لا تتغير و لا تتعلق بمعيار إنشاء الزوج $< a , b >$, و بحيث يتم البحث عن بديل في حال إخفاق أحد الطرفين , تؤدي إلى تبسيط المسألة , و لكن قد تشوه الواقع , و تدعى عملية تجريد .
يمكن البحث عن حلول لهذه المسألة على الشكل التالي : نجرب و بالتتالي توحيد عنصرين من المجموعتين A و B في زوج إلى أن تنتهي عناصر المجموعتين . لنحاول إيجاد جميع التوزيعات المستقرة التي تمكننا بسهولة من العثور على الحل .

تعقيد الخوارزميات – COMPLEXITY

يلزم حساب تعقيد الخوارزميات للتمييز بين خوارزميات مختلفة حيث جميعها تحل نفس المشكلة (أي لتحديد الخوارزمية الأسهل).
بالإضافة إلى ذلك فإن حساب درجة التعقيد هام جداً لقياس زمن تنفيذ برنامج ما.
قبل أن نبدأ بدراسة تعقيد الخوارزميات , لا بد من ذكر الملاحظات المهمة التالية :

- بعض المفاهيم الرياضية اللازمة لمقارنة الخوارزميات:

- نقول عن التابع f إنه مُسيطر عليه من قبل g و نرمز إلى ذلك بـ $f = O(g)$ أو $f(n) = O(g(n))$ إذا كان :

$$f(n) = O(g(n)) \quad \Leftrightarrow \quad \exists\ c,N \mid n > N \quad \Rightarrow \quad f(n) \le c\ g(n)$$

مثال-1:

$$\left.\begin{array}{l} f(n) = n \\ g(n) = 3n \end{array}\right\} \Rightarrow \qquad n < c(3n) \quad \Rightarrow \quad c > \frac{1}{3} \quad \Rightarrow \quad \{\ f(n) = O(g(n))\ ;\ c > \frac{1}{3}\ \}$$

مثال-2:

$$f(n) = 5n^3 = O(n^3) \quad ; \quad f(n) = n^2 + n = O(n^2) \quad ; \quad 6n^2 + 3n = O(n^2) \quad ; c=7\ ,\ n=3$$

- للمقارنة نعتمد على الحقائق التالية :

$$O(1) \le O(\log n) \le O(n^k) \le O(n) \le O(n \log n) \le O(n^c) \le O(c^n) \le O(n!) \quad ;(0\le k\le 1\le c)$$

- $$q + 2q + 3q + ... + nq = \frac{n(n+1)q}{2} \quad ; \quad 1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$$

$$1 + q + q^2 + ... + q^{n-1} = \frac{1-q^n}{1-q} \quad ; \quad 1 + q + q^2 + q^3 + ... + q^n = \frac{q^{n+1}-1}{q-1}$$

ملاحظات:

- الضرب بثابت : $c*O(g) = O(g)$
- جمع و ضرب التوابع : $O(g1) + O(g2) = O(Max\ \{\ g1\ ,\ g2\ \})$
- جمع و ضرب التوابع : $O(g1) * O(g2) = O(\ g1 * g2\ )$
- جمع و ضرب التوابع : $O(g1) + O(g2) = O(Max\ \{\ g1\ ,\ g2\ \})$

درجة تعقيد خوارزمية:

درجة تعقيد مسألة هي درجة تعقيد أفضل خوارزمية لحل هذه المسألة .

- حساب زمن تنفيذ خوارزمية :

يتعلق زمن تنفيذ برنامج ما بالعوامل التالية:

1- المعطيات الحالية للبرنامج
2- نوعية code البرنامج المولد من قبل المترجم compiler (تميز النوعية حسب اللغة المستخدمة)
3- نوعية و جودة المكتبات المستخدمة
4- سرعة الخوارزمية
5- جودة البرنامج المكتوب من قبل المبرمج

لحساب زمن التنفيذ يجب تحديد أمرين و هما :

**1) التوصيف الكمي للمسألة أي بُعد (أو طول أو حجم) معطيات المسألة : فمثلاً

بُعد مسألة ترتيب عناصر sort = عدد العناصر اللازم ترتيبها

بُعد مسألة معالجة كثير حدود = درجة كثير الحدود

بُعد مسألة بيان graph = عدد الرؤوس أو عدد الأضلاع

بُعد مسألة مصفوفات من المرتبة (m x n) = مثلاً max(m,n) أو m+n أو m x n

**2) تحديد العمليات الأساسية التي سوف تأخذ بعين الاعتبار (حيث نهمل العمليات البسيطة أمام العمليات التي هي أكثر كلفة) فمثلاً:

- بالنسبة لعدد من العمليات المتتالية فإن زمن الحساب يجمع جمعاً
- بالنسبة لعمليات الشرطية من الشكل if (c) a; else b; فإن الزمن يحسب بالعلاقة T(c) + Max(T(a),T(b))
- بالنسبة لحلقة تنفذ n مرة على الأكثر و باعتبار m زمن تنفيذ ما داخل الحلقة فيكون زمن التنفيذ هو n x m
- بالنسبة لطلب برنامج جزئي غير عودي تحسب درجة تعقيد الإجرائية باستخدام نفس القواعد
- بالنسبة لبرنامج جزئي عودي نستخدم علاقته العودية لحساب درجة التعقيد , فمثلاً في خوارزمية حساب العاملي لعدد صحيح موجب نعتبر أن عملية الضرب هي العملية الأساسية و بالتالي نلاحظ:

$T(0) = 0$

$T(n) = 1 + T(n-1)$; $n >= 1$ => $T(n) = n$ الحل هو

قياس درجة التعقيد:

يتعلق زمن خوارزمية بالمعطيات التي تعالجها (أي حسب حجمها و محتوياتها) فمثلاً يوجد فرق بين مسألة البحث عن عنصر في قائمة عناصرها أعداد صحيحة و قائمة أخرى عناصرها سجلات أو حتى قوائم مترابطة ....

بفرض D_n معطيات المسألة ذات الحجم n نميز بين ما يلي :

1- التعقيد الزمني في أحسن الأحوال للخوارزمية A :

$Min_A(n) = Min \{ Cost_A(d) \ ; \ d \in D_n \}$; حيث cost هو كلفة الزمن اللازم

2- التعقيد الزمني في أسوأ الأحوال للخوارزمية A :

$Max_A(n) = Max \{ Cost_A(d) \ ; \ d \in D_n \}$

3- التعقيد الزمني الوسطي للخوارزمية A :

$$Average_A(n) = \sum_{d \in D_n} P(d) \cdot Cost_A(d)$$

حيث p احتمال أن تكون معطيات الخوارزمية هي d .

مثال: حساب التعقيد الزمني الوسطي لمسألة البحث التسلسلي عن العنصر x ضمن قائمة تحوي n عنصراً .

الحل: لنسمي خوارزمية البحث A .

من الواضح أن أسوأ الحالات الممكنة هي وجود العنصر المطلوب في نهاية القائمة و بالتالي : $Max_A(n) = n$

بينما أحسن حالة هي كون العنصر المطلوب هو أول عنصر في القائمة أي نقوم بعملية مقارنة واحدة فقط : $Min_A(n) = 1$

أما لحساب التعقيد الزمني الوسطي نحدد ما يلي:

مع الأخذ بعين الاعتبار أن العنصر x يمكن أن يكون موجوداً في القائمة و يمكن لا نفرض أن q هو احتمال وجود x في القائمة و لنفرض أنه إذا وجد x في القائمة فإن المواضع كلها متساوية الاحتمال. فمن أجل $0 \le i \le n-1$ نرمز بـ $D_{n,i}$ إلى مجموعة كل القوائم التي طولها n و التي يظهر فيها x . و بالتالي:

$P(D_{n,-1}) = 1 - q$ => $Cost(D_{n,-1}) = n$

$P(D_{n,i}) = q / n$ => $Cost(D_{n,i}) = i+1$

$$Average_A(n) = \frac{q}{n}\sum_{i=0}^{n-1}(i+1) + (1-q)\cdot n = \frac{q}{n}\frac{n(n+1)}{2} + (1-q)\cdot n = \frac{q(n+1)}{2} + (1-q)\cdot n$$

فإذا كنا نعلم سلفاً أن x موجود في القائمة فإن q=1 و بالتالي:

$$Average_A(n) = \frac{(n+1)}{2}$$

و إذا كان احتمال وجود x في القائمة q=1/2 فإن:

$$Average_A(n) = \frac{3n+1}{4}$$

أمثلة:

مثال 1*) حساب درجة التعقيد لخوارزمية الإدخال لترتيب صف (insert) :

$$Cost(n) = 1+2+3+.....+n = \frac{n(n+1)}{2} = \frac{n^2}{2}+\frac{n}{2} \approx \frac{n^2}{2} = O(n^2)$$

ملاحظة: نهمل الثوابت في درجة التعقيد .

مثال 2*) حساب درجة التعقيد لخوارزمية الفرز بالفقاعات (Bubble Sort) :

بُعد المسألة هو n و يحتوي البرنامج على حلقتين (واحدة داخل الأخرى), هناك n-1 مروراً في الحلقة الخارجية. في المرور i نجري i عملية مقارنة و i عملية تبديل مواقع و لذالك:

$$Cost(n) = \sum_{i=1}^{n-1} 2i = 2\frac{n(n-1)}{2} = n(n-1) = O(n^2)$$

مثال 3*) حساب درجة التعقيد لخوارزمية البحث الثنائي عن عنصر في قائمة (Binary Search) :

تعتمد الخوارزمية على حساب منتصف المجموعة و اختبار وجود العنصر في المنتصف أو في القسم الأيسر أو في القسم الأيمن ثم تعيد تقسيم المجموعة المتبقية .... حتى نصل إلى الجواب. إذاً في كل مرحلة ينقص عدد العناصر إلى النصف و نقوم بعمليتين : مقارنة و إعادة تقسيم :

$T(1) = 1$

$T(n) = 1 + T(\frac{n}{2})$

$T(n) = 1 + 1 + T(\frac{n}{4})$

$T(n) = 2 + T(\frac{n}{4})$

$T(n) = 2 + 1 + T(\frac{n}{8})$

$T(n) = 3 + T(\frac{n}{2^3})$

.....

$T(n) = k + T(\frac{n}{2^k})$

بفرض : $n = 2^k$ => $k = \log n$

$T(n) = \log n + T(\frac{n}{n})$

$T(n) = \log n + T(1)$

$T(n) = \log n + 1$

$T(n) = O(\log n)$

ملاحظة: خوارزمية البحث الثنائي هي أسرع خوارزمية بحث. بينما بالنسبة للترتيب فإن خوارزمية الترتيب السريع quick sort هي الأسرع حيث $Cost(n) = n \log(n)$.

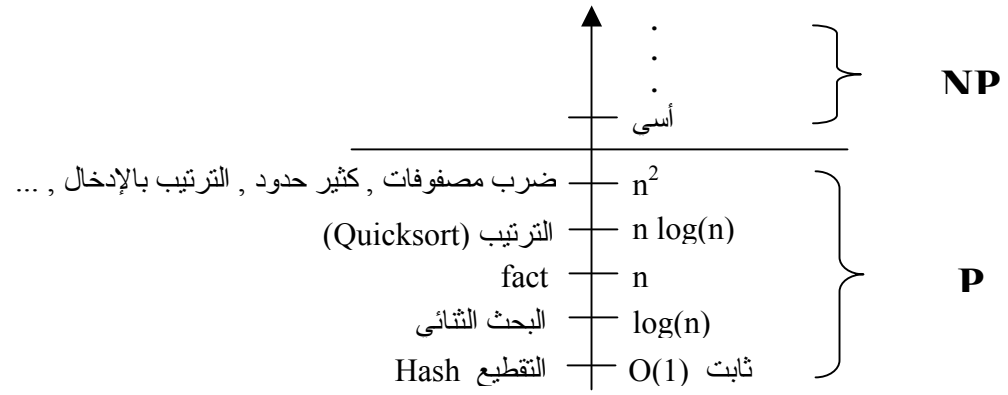
- المسائل NP (Non Polynomial) and P (Polynomial) :

نقول عن خوارزمية أنها **فعالة** (P = polynomial) إذا كانت درجة تعقيدها ذات درجة كثير حدود .

و نقول عن خوارزمية أنها **غير فعالة** (NP = non polynomial) إذا كانت درجة تعقيدها ليست بدرجة كثير حدود , أي أسية .

إن المسائل التي تنتمي إلى P هي مسائل سهلة و قابلة للمعالجة بينما المسائل التي تنتمي إلى NP هي مسائل صعبة و غير قابلة للمعالجة إلا في حالة معطيات ذات حجم محدود .

يبين الشكل التالي أمثلة على درجات تعقيد مختلفة مع انتمائهم إلى P أو NP :



أمثلة على حساب التعقيد:

✓ **احسب درجة تعقيد الخوارزمية العودية لحساب العاملي لعدد صحيح موجب :**

نعتبر أن عملية الضرب هي العملية الأساسية و بالتالي :

$T(0) = 0$
$T(n) = 1 + T(n-1)$
$= 2 + T(n-2)$
$= 3 + T(n-3)$
.....
$= k + T(n-k)$; $n - k = 0 \Rightarrow k = n \Rightarrow$
$= n + T(0)$
$= n + 0$
$T(n) = O(n)$

✓ **احسب درجة تعقيد الخوارزمية العودية لحساب عدد مرفوع إلى قوة :**

نعتبر أن عملية الضرب هي العملية الأساسية و بالتالي :

$T(0) = 0$
$T(n) = 1 + T(n-1)$
$= 2 + T(n-2)$
$= 3 + T(n-3)$
.....
$= k + T(n-k)$; $n - k = 0 \Rightarrow k = n \Rightarrow$
$= n + T(0)$
$= n + 0$
$T(n) = O(n)$

✓ **احسب درجة تعقيد الخوارزمية العودية لحساب ناتج ضرب عددين صحيحين :**

نعتبر أن عملية الجمع هي العملية الأساسية و بالتالي :

$T(0) = 0$
$T(n) = 1 + T(n-1)$
$= 2 + T(n-2)$
.....
$= k + T(n-k)$; $n - k = 0 \Rightarrow k = n \Rightarrow$
$= n + 0$
$T(n) = O(n)$

✓ **احسب درجة تعقيد خوارزمية الفرز بالفقاعات (Bubblesort) :**

نعتبر أن عملية المقارنة هي العملية الأساسية (حيث أن الخوارزمية تكرارية) و بالتالي :

$T(n) = (n-1) + (n-2) + ... + 2 + 1$
$= \frac{n(n-1)}{2}$
$T(n) = O(n^2)$

✓ **احسب درجة تعقيد خوارزمية الفرز بالإدخال (Insertsort) :**

خوارزمية الترتيب بالإدخال (Insertsort) تعتمد على الفكرة التالية :
نبدأ من عند العنصر الثاني , حيث نعتبر العنصر الأول مرتباً, و نحاول حشر العنصر الثاني بين العناصر ما قبله (أي الأول) , فإذا كان الأول أكبر نبدل . الآن نبدأ من عند العنصر الثالث و نحاول حشره ما بين العنصرين المرتبين ما قبله , مجرد ما نوصل إلى أول عنصر أصغر , نتوقف , لأن أصبح الترتيب صحيحاً . و هكذا نكرر نفس الخطوات من عند العنصر الرابع , الخامس ... إلى العنصر الأخير . ففي أول مرة نقارن مرة واحدة فقط , في المرة الثانية مرتين , في المرة الثالثة ثلاثة مرات , ... , و في المرة الأخيرة n-1 مرة . إذاً :
نعتبر أن عملية المقارنة هي العملية الأساسية (حيث أن الخوارزمية تكرارية) و بالتالي :

$$T(n) = 1 + 2 + 3 + \ldots\ldots + (n-1)$$
$$= \frac{n(n-1)}{2} = O(n^2)$$

✓ **احسب درجة تعقيد خوارزمية الفرز السريع (Quicksort) :**

خوارزمية الترتيب السريع (Quicksort) تعتمد على الفكرة التالية (تشبه خوارزمية الدمج) :
نبحث عن القيمة الوسطية ما بين الـ n عنصر و نجزئ عناصر الصف إلى صفين جزئيين : الأول نضع فيه جميع العناصر الأصغر من القيمة الوسطى و الثاني نضع فيه جميع العناصر الأكبر . فلحساب التعقيد نفترض أن تقريباً نصف العناصر أصغر و النصف الثاني أكبر . إذاً من أجل كل عنصر لدينا مقارنتين : أصغر أم أكبر . الآن نعيد نفس الخطوات على الصفوف الجزئية ... حتى أن نحصل على صفوف مؤلفة من عنصر وحيد . الآن ندمج كل صفين معاً (دون مقارنة , علماً أنه دوماً لدينا القيم الأصغر على اليسار و القيم الأكبر على اليمين) . إذاً : نعتبر أن عملية المقارنة هي العملية الأساسية (حيث أن الخوارزمية عودية) و بالتالي :

$$T(1) = 1$$
$$T(n) = n + n + 2\,T(\frac{n}{2})$$
$$= 2n + 2\,T(\frac{n}{2})$$
$$= 4n + 4\,T(\frac{n}{4})$$
$$= 6n + 8\,T(\frac{n}{8}) \quad \ldots\ldots\ldots\ldots\ldots\ldots$$
$$= 2kn + 2^k\,T(\frac{n}{2^k}) \quad ; \quad \frac{n}{2^k} = 1 \quad \Rightarrow \quad k = \log(n) \quad \Rightarrow$$
$$= 2\,n\log(n) + n$$
$$T(n) = O(n\log(n))$$

✓ **احسب درجة تعقيد خوارزمية هانوي (Hanoi) :**

نعتبر أن عملية النقل (= طباعة) هي العملية الأساسية و بالتالي :

$$T(1) = 1$$
$$T(n) = 1 + 2\,T(n-1)$$
$$= 1 + 2 + 4\,T(n-2)$$
$$= 1 + 2 + 4 + 8\,T(n-3)$$
$$= 1 + 2 + 4 + 8 + 16\,T(n-4) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$= \frac{1-2^k}{1-2} + 2^k\,T(n-k) \quad ; \quad n - k = 1 \quad \Rightarrow \quad k = n - 1 \quad \Rightarrow$$
$$= \frac{1-2^{n-1}}{1-2} + 2^{n-1}\,T(1)$$
$$= 2^{n-1} - 1 + 2^{n-1}$$
$$= 2 \cdot 2^{n-1} - 1$$
$$= 2^n - 1$$
$$T(n) = O(2^n) \quad \Rightarrow \quad \text{NP} \quad \text{الخوارزمية غير فعالة}$$

✓ **احسب درجة تعقيد خوارزمية فيبوناتشي (Fibonacci) :**

نعتبر أن عملية الجمع هي العملية الأساسية (حيث أن الخوارزمية عودية) و بالتالي :

$T(0) = 0$

$T(1) = 0$

$T(n) = 1 + T(n-1) + T(n-2)$

$= 1 + 2 + T(n-2) + 2T(n-3) + T(n-4)$

$= 1 + 2 + 4 + T(n-3) + 3T(n-4) + 3T(n-5) + T(n-6)$

$= 1 + 2 + 4 + 8 + T(n-4) + 4T(n-5) + 6T(n-6) + 4T(n-7) + T(n-8)$

$= 1 + 2 + 4 + 8 + 16\ T(n-4)$ (وسطياً)

$= \frac{1-2^k}{1-2} + 2^k\ T(n-k)$; $n - k = 0 \Rightarrow k = n \Rightarrow$

$= 2^n - 1 + 2^n \cdot 0$

$T(n) = O(2^n) \Rightarrow$ NP الخوارزمية غير فعالة

زمن تنفيذ خوارزمية بدلالة التعقيد و حجم المعطيات : (للإطلاع)

حجم المعطيات						
1.000.000	100.000	10.000	1.000	100	10	التعقيد
10^{-6} s	10^{-6} s	10^{-6} s	10^{-6} s	10^{-6} s	10^{-6} s	1
19.93 ns	16.61 ns	13.29 ns	9.97 ns	6.64 ns	3.32 ns	$\log_2 n$
1000 ms	100 ms	10 ms	1 ms	0.1 ms	0.01 ms	N
19931.57 ms	1660.96 ms	132.88 ms	9.97 ms	0.66 ms	0.03 ms	n log n
11.57 d	2.78 h	100.00 s	1.00 s	0.01 s	0.0001 ms	N^2
31710 y	31.71 y	11.57 d	1000.00 s	1.00 s	0.001 ms	N^3
∞	∞	∞	1×10^{295} y	4.02×10^{16} y	1.02 s	2^n

(ns = nano second , ms = milli second , s = second , min = minute , h = hour , d = day , y = year)

حجم المعطيات التي يمكن معالجتها بدلالة التعقيد الزمني و زمن التنفيذ : (للإطلاع)

زمن التنفيذ				
1 y	1 h	1 min	1 s	التعقيد
∞	∞	∞	∞	1
∞	∞	∞	∞	$\log_2 n$
86×10^9	36×10^8	6×10^7	10^6	N
97×10^8	13×10^7	28×10^5	63×10^3	n log n
26×10^4	60×10^3	7.7×10^3	10^3	n^2
4400	1500	360	100	n^3
36	31	25	19	2^n

من دراسة الجدولين السابقين يتبين أن بعض الخوارزميات لا تصلح , و يجب عدم توظيفها لحل المسائل باستعمال الحاسوب .

☙ Advanced Data Structures - بنى معطيات متقدمة ❧

سوف نتعرف في الدروس القادمة على أهم بنى معطيات متقدمة و التي هي :

- بنى المعطيات الخطية :
  - الأشعة Arrays
  - السلاسل الخطية Linked Lists
  - المكدسات Stacks
  - الأرتال Queues
- البنى الشجرية Trees
- البيانات Graphs

بنى المعطيات الخطية :

تُعد البنى الخطية أكثر البنى استعمالاً لتنظيم المعطيات , لأنها تسمح بتنظيم المعطيات التي نريد معالجتها تنظيماً تسلسلياً . من أهم بنى المعطيات الخطية لدينا :

الأشعة - Arrays

من المعروف أن الشعاع صف أحادي البعد حيث نذكر إنه يمكن الوصول إلى أي عنصر من شعاع وصولاً مباشراً .

السلاسل الخطية - Linked Lists

يمكن تمثيل السلاسل الخطية إما باستخدام الأشعة أو بالمؤشرات . حيث يمكننا في الحالتين تطبيق العمليات التالية عليها : إيجاد طول السلسلة , قراءة السلسلة (تصاعدياً و تنازلياً) , بحث عن عنصر , حشر عنصر و حذف عنصر .

1- تمثيل السلاسل الخطية باستخدام الأشعة:

و يتم ذلك بتعريف سجل مؤلف من حقلين : حقل طول السلسلة و حقل صف عناصر السلسلة . *مثال*:

```
const lmax=100;
struct listType {
       int  length;
       elem val[lmax];     //  elem = أي نمط معطيات معروف
};
```

سلبيات تمثيل السلاسل باستخدام الأشعة :

إن الطول الأعظم للسلسلة محدود , فلا يمكن إضافة عناصر أكثر من طول السلسلة . لذلك يجب أخذ هذه الفكرة بعين الاعتبار أثناء تعريف السلسلة و حجز طول زائد حتى لا يحدث خطأ . بالإضافة إلى ذلك فأن عمليات الحشر و الحذف تسبب عدداً كبيراً من الإزاحات في الذاكرة .

ملاحظات :

L.length=0 ⇔ L فارغة

L.val[i] = العنصر ذا الرقم i من السلسلة L

إضافة عنصر x في الموقع k من السلسلة الخطية

```
listType insert(listType L,elem x,int k)
{ if ((L.length<lmax)&&(k<=L.length))
     {
       for (int i=L.length-1;i>=k;i--)
           L.val[i+1]=L.val[i];
       L.val[k]=x;
       ++L.length;
     }
  return L;
}
```

حذف عنصر x من الموقع k في السلسلة الخطية

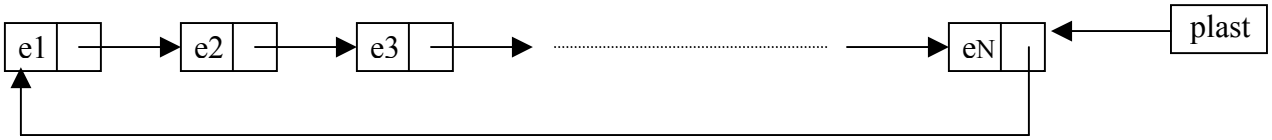
```
listType delet(listType L,int k)
{
  if ((k<L.length)&&(k>=0))
     {
       for (int i=k;i<L.length-1;i++)
           L.val[i]=L.val[i+1];
       --L.length;
     }
  return L;
}
```

2- تمثيل السلاسل الخطية باستخدام المؤشرات:

بما أننا درسنا هذا الموضوع بشكل معمق في السنة الدراسية الأولى و بالإضافة إلى المراجعة في بداية السنة الثانية , سنكتفي بالكلام عنها , و لكن يوجد مفهوم جديد يجب التعرف عليه , و هو :

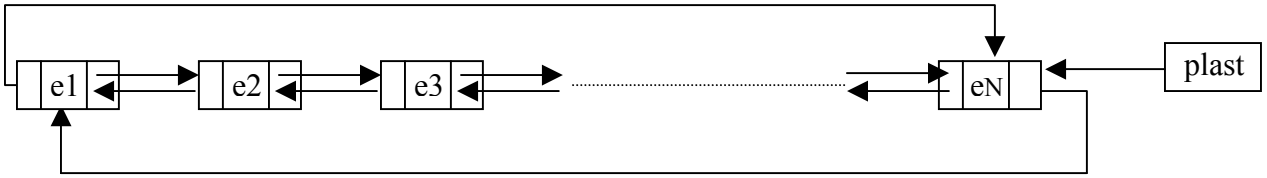
السلاسل الدائرية :

بدلاً من أن يأخذ المؤشر في آخر عنصر في السلسلة القيمة nil نجعل هذا المؤشر يؤشر إلى أول عنصر في السلسلة . و نحدد السلسلة بمؤشر إلى آخر عنصر فيها , كما في الشكل التالي :



السلاسل مضاعفة الارتباط :

نجعل فيها مؤشر العنصر الأخير أن يؤشر إلى أول عنصر , كما أننا نجعل مؤشر العنصر الأول أن يؤشر إلى آخر عنصر .



تمرين: مثلث باسكال - Pascal Triangle

يُمثل مثلث باسكال بالشكل التالي (حيث عدد الأسطر هنا = 6) :

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
```

و المطلوب :

نريد حساب و طباعة مثلث باسكال من السطر 1 حتى السطر n باستخدام قائمة مترابطة واحدة فقط (نمثل فيها العناصر المختلفة عن الواحد فقط) . بعد حساب السطر i فإن القائمة المترابطة تحوي فقط عناصر السطر i (المختلفة عن الواحد) . من الطبيعي أن حساب عناصر السطر i يعتمد على عناصر السطر i - 1 وفق القاعدة المعروفة : t[i,j] = t[i-1,j] + t[i-1,j-1]

حيث j هو موقع العنصر في السطر i من المثلث t .

1- حدد بنى معطيات مناسبة لتمثيل عناصر مثلث باسكال .

2- اكتب إجرائية تقوم بحساب عناصر السطر i و تخزينها في القائمة المترابطة (لاحظ أن نفس القائمة تحوي عناصر السطر السابق i - 1 قبل استدعاء هذه الإجرائية) .

3- اكتب إجرائية تقوم بطباعة أول n سطراً من مثلث باسكال وفق الشكل النظامي (كما في الشكل) .

الحل:

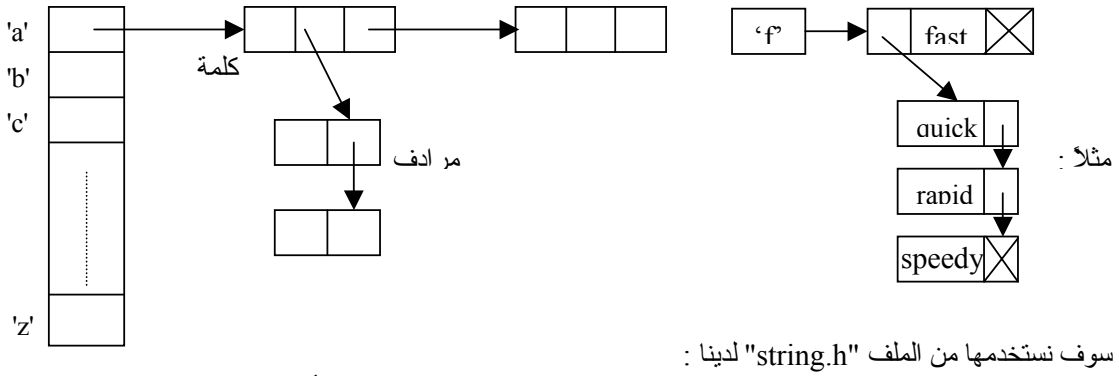
```
#include <iostream.h>
#include <conio.h>
struct str{
       int  val;
       str *prev,*next;
  };
str *pfirst=0,*plast=0;
str s;
int n;
void init();
void print();
/*****************************************************************************/
void main()
{
```

```
  cout<<"Enter n = "; cin>>n;
  cout<<"\n\n";
  if (n==1) cout<<"1 \n";
  else
  if (n==2)
     {
       cout<<"1  \n";
       cout<<"1  1 \n";
     }
  else
     {
       cout<<"1  \n";
       cout<<"1  1 \n";
       for (int i=3;i<=n;i++)
           { init();
             print();
           }
     }
  getch();
}
/*****************************************************************************************/
void init()
{
  if (plast==0)
     {
      s.next=0;
      s.prev=0;
      s.val=2;
      plast=new str(s);
      pfirst=plast;
     }
  else
     {
      s.next=0;
      s.prev=plast;
      s.val=1;
      plast->next=new str(s);
      plast=plast->next;
      str *p=plast;
      while (p!=pfirst)
        { p->val += (p->prev)->val;
          p=p->prev;
        }
      p->val += 1;
     }
}
/*****************************************************************************************/
void print()
{
  str *p=pfirst;
  cout<<"1  ";
  while (p!=0)
    {

      cout<<p->val<<"  ";
      p=p->next;
    }
  cout<<"1"<<endl;
```

```
}
```

مسألة بناء مكنز (قاموس مترادفات) - Thesaurus

استخدم السلاسل الخطية لبناء مكنز (قاموس مترادفات – Thesaurus) , حيث أنه يحوي مجموعة من الكلمات مع مترادفاتها, بحيث تكون الكلمات و المترادفات مرتبة أبجدياً . لتحقيق ذلك نفترض أنه لدينا شعاع من المؤشرات مرقم حسب الأحرف الأبجدية. كل مؤشر يؤشر إلى سلسلة خطية من الكلمات التي تبدأ بنفس الحرف الأبجدي . كل كلمة بدورها تحوي مؤشراً آخر إلى سلسلة خطية تحوي مترادفات هذه الكلمة مرتبة أبجدياً كما يوضح الشكل. و المطلوب كتابة الخوارزميات التالية: 1- إضافة كلمة 2- إضافة مرادف لكلمة موجودة 3- حذف كلمة 4- حذف مرادف 5- البحث عن كلمة 6- البحث عن مرادف 7- طباعة مترادفات كلمة معينة 8- طباعة محتويات المكنز



الحل:

من أهم التوابع التي سوف نستخدمها من الملف "string.h" لدينا :

`strcpy(char s1[],char s2[])` : الذي يقوم بإسناد (نسخ) محتوى السلسلة الثانية إلى الأولى

`stricmp(char s1[],char s2[])` : الذي يقارن ما بين سلسلتين , أيهما الأصغر حسب الأحرف الأبجدية , فهو يمكن أن يعيد ثلاثة حالات : `(if s1==s2) "==0" || "<0" if (s1<s2) || ">0" if (s1>s2)`

`strlwr(char s[])` : لتحويل جميع أحرف السلسلة إلى أحرف صغيرة

سوف نعرف على توابع البحث بشكل مناسب , حتى نستفيد منهما في بقية الإجرائيات , بالشكل التالي :

```
char searchWord(words* pfirst,char word[],words* &pw1,words* &pw2);
```

حيث نمرر pfirst الذي هو مؤشر بداية السلسلة المقابلة للكلمة , و الكلمة المطلوبة , و مؤشرين : الأول يرجع عنوان الكلمة في حال تم إيجادها(نستفيد منه في الحشر و الحذف و طباعة المترادفات) , أما المؤشر الثاني يرجع العنوان ما قبل الكلمة المطلوبة (نستفيد منه في الحذف)

```
char searchSyn(synonym* pfirst,char syn[],synonym* &ps1,synonym* &ps2);
```

حيث نمرر pfirst الذي هو مؤشر بداية السلسلة المقابلة للمرادف , و المرادف المطلوب , و مؤشرين : الأول يرجع عنوان المرادف في حال تم إيجاده(نستفيد منه في الحشر و الحذف) , أما المؤشر الثاني يرجع العنوان ما قبل المرادف المطلوب (نستفيد منه في الحذف)

الآن ممكن أن نستفيد من هذين التابعين : مثلاً - قبل أن نحشر كلمة جديدة , يجب أن نتأكد من عدم وجودها

- قبل أن نحشر مرادف لكلمة يجب أن نتأكد من وجود الكلمة و عدم وجود المرادف
- قبل حذف كلمة أو مرادف يجب أن نتأكد من وجودهما , هكذا ...

```
#include <iostream.h>
#include <conio.h>      //   -> clrscr() , getch() , getche()
#include <string.h>  // -> strlen() , strcpy(destination,source)=copy string ,
                     //  -> stricmp(s1,s2) , strlwr(string)
const L=30;         //  L= maxlength of a word or a synonym
const n=26;        // there are 26 alphabetic letters ( in English )
struct synonym { //  = muradef
       char     syn[L];
       synonym* next;
};
struct words {
       char     word[L];
       words*   next;
       synonym* down;
};
typedef words* dictionary[n];
dictionary d;
words *pw1,*pw2;
synonym *ps1,*ps2;
char c;  //  = choice of user
```

```
char menu();
char searchWord(words* pfirst,char word[],words* &pw1,words* &pw2);  // pw1=address
                                        // of word , pw2=address of previous word
char searchSyn(synonym* pfirst,char syn[],synonym* &ps1,synonym* &ps2);
                      // ps1=address of synonym , ps2=address of previous synonym
void insertWord(words* pw1,words* &pw2,char word[]);
void insertSyn(synonym* ps1,synonym* &ps2,char syn[]);
void deleteWord(words* pw1,words* &pw2,char word[]);
void deleteSyn(synonym* ps1,synonym* &ps2,char syn[]);
void printSynonyms(synonym *ps);    // prints  all synonyms of one word
void printDictionary(dictionary d); //  prints the whole dictionary
/*****************************************************************************/
void main()
{
  do {
       char word[L],syn[L];
       c=menu();
       switch (c){
       case '1':{
                  cout<<"Enter new word = ";
                  cin>>word; strlwr(word);
                  if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='y')
                                          // ASCII : 'a' - 'z'  ( 97 - 122 )
                      cout<<"The word \""<<word<<"\" does already exist ! \n";
                  else
                      insertWord(pw1,pw2,word);
                  break;
                }
       case '2':{
                  cout<<"Enter basic word of synonym = ";
                  cin>>word; strlwr(word);
                  cout<<"Enter synonym of "<<word<<" = ";
                  cin>>syn;  strlwr(syn);
                  if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='n')
                      cout<<"Sorry , the word \""<<word<<"\" doesn't exist ! \n";
                  else
                     { char v= searchSyn(pw1->down,syn,ps1,ps2);
                       if ((pw1->down!=0)&&(v=='y'))
                           cout<<"The synonym \""<<syn<<"\" does already exist!\n";
                       else
                           insertSyn(ps1,ps2,syn);
                     }
                  break;
                 }
       case '3':{
                  cout<<"Enter the word you want to delete = ";
                  cin>>word; strlwr(word);
                  if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='n')
                      cout<<"Sorry , the word \""<<word<<"\" does not exist ! \n";
                  else
                      deleteWord(pw1,pw2,word);
                  break;
                 }
       case '4':{
                  cout<<"Enter the word you want to delete its synonym = ";
                  cin>>word; strlwr(word);
                  cout<<"Enter the synonym you want to delete = ";
                  cin>>syn;  strlwr(syn);
                  if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='n')
                      cout<<"Sorry , the word \""<<word<<"\" does not exist ! \n";
                  else
                  if (searchSyn(pw1->down,syn,ps1,ps2)=='n')
                      cout<<"Sorry , the synonym \""<<syn<<"\" does not exist ! \n";
```

```
                else
                    deleteSyn(ps1,ps2,syn);
                break;
               }
        case '5':{
                cout<<"Enter the word you want to search for = ";
                cin>>word; strlwr(word);
                if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='y')
                    cout<<"\""<<word<<"\" is found ! \n";
                else
                    cout<<"\""<<word<<"\" is not found ! \n";
                break;
               }
        case '6':{
                cout<<"Enter the basic word of the synonym = ";
                cin>>word; strlwr(word);
                cout<<"Enter the synonym you want to search for = ";
                cin>>syn;  strlwr(syn);
                if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='n')
                    cout<<"Sorry , the word \""<<word<<"\" is not found ! \n";
                else
                if (searchSyn(pw1->down,syn,ps1,ps2)=='y')
                    cout<<"\""<<syn<<"\" is found ! \n";
                else
                    cout<<"\""<<syn<<"\" is not found ! \n";
                break;
               }
        case '7':{
                cout<<"Enter the word you want to see its synonyms = ";
                cin>>word; strlwr(word);
                if (searchWord(d[int(word[0])-97],word,pw1,pw2)=='y')
                    if (pw1->down==0)
                        cout<<"Sorry , there are no synonyms for this word ! \n";
                    else
                        printSynonyms(pw1->down);
                else
                    cout<<"Sorry , this word doesn't exist ! \n";
                break;
               }
        case '8':{
                printDictionary(d);
                break;
               }
        }  // end of switch
        getch();
  } while (c!='9');
}
/*****************************************************************************************/
char menu()
{
  char c;
  do {
        clrscr();
        cout<<"1= Insert New Word \n";
        cout<<"2= Insert New Synonym \n";
        cout<<"3= Delete Word \n";
        cout<<"4= Delete Synonym \n";
        cout<<"5= Search For a Word \n";
        cout<<"6= Search For a Synonym \n";
        cout<<"7= Print All Synonyms Of a Word \n";
        cout<<"8= Print The Whole Dictionary \n";
        cout<<"9= Exit \n\n\n";
        cout<<"Enter your choice : ";
```

```
       c=getche();   //  getche() : reads a char and print it on userscreen
       cout<<"\n\n\n";
  }  while ((int(c)<49)||(int(c)>57));  // ASCCI : '1' - '9'  ( 49 - 57 )
  return c;
}
/*****************************************************************************/
char searchWord(words* pfirst,char word[],words* &pw1,words* &pw2)
                                // pw1=address of word pw2=address of previous word
{
  pw1=pfirst;
  pw2=pfirst;
  while ((pw1!=0)&&(stricmp(pw1->word,word)<0))
     {
       pw2=pw1; pw1=pw1->next;
     }
  if ((pw1!=0)&&(stricmp(pw1->word,word)==0))
       return 'y';
  else
       return 'n';
}
/*****************************************************************************/
char searchSyn(synonym* pfirst,char syn[],synonym* &ps1,synonym* &ps2)
                        // ps1=address of synonym , ps2=address of previous synonym
{
  ps1=pfirst; ps2=pfirst;
  while ((ps1!=0)&&(stricmp(ps1->syn,syn)<0))
    {
      ps2=ps1; ps1=ps1->next;
    }
  if ((ps1!=0)&&(strcmp(ps1->syn,syn)==0))
      return 'y';
  else
      return 'n';
}
/*****************************************************************************/
void insertWord(words* pw1,words* &pw2,char word[])
{
  words* p=new words();
  strcpy(p->word,word);
  p->next=pw1;
  p->down=0;
  if ((pw1==0)&&(pw2==0))      //  linked list is empty
       d[int(word[0])-97]=p;
  else
  if (pw1==pw2)               //  insert at the beginning
       d[int(word[0])-97]=p;
  else                       //  insert anywhere else
       pw2->next=p;
}
/*****************************************************************************/
void insertSyn(synonym* ps1,synonym* &ps2,char syn[])
{
  synonym* p=new synonym();
  strcpy(p->syn,syn);
  p->next=ps1;
  if ((ps1==0)&&(ps2==0))   //  there are no synonyms for this word
      pw1->down=p;
  else
  if (ps1==ps2)  //  insert at the beginning
      pw1->down=p;
  else                  //  insert anywhere else
      ps2->next=p;
}
```

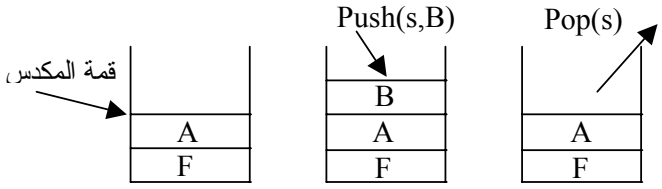
```
/*****************************************************************************/
void deleteWord(words* pw1,words* &pw2,char word[])
{
  synonym *ps=pw1->down,*aux;
  while (ps!=0)     //  first: delete all synonyms of this word
    { aux=ps;
      ps=ps->next;
      delete aux;
    }
  if (pw1==pw2)  //  delete first word
    {
      d[int(word[0])-97]=pw1->next;
      delete pw1;
    }
  else                          //  delete anywhere else
      {
        pw2->next=pw1->next;
        delete pw1;
      }
}
/*****************************************************************************/
void deleteSyn(synonym* ps1,synonym* &ps2,char syn[])
{
  if (ps1==ps2)            //  delete first synonym
      pw1->down=ps1->next;
  else                     //  delete anywhere else
      ps2->next=ps1->next;
  delete ps1;
}
/*****************************************************************************/
void printSynonyms(synonym *ps)    // prints  all synonyms of one word
{
  while (ps!=0)
    {
      cout<<ps->syn<<" , ";
      ps=ps->next;
    }
}
/*****************************************************************************/
void printDictionary(dictionary d) //  prints the whole dictionary
{
  words *pw;
  synonym *ps;
  for (int i=0;i<=25;i++)
      {
        pw=d[i];
        while (pw!=0)
          {
            cout<<"\n Word[ "<<pw->word<<" ] = ";
            ps=pw->down;
            while (ps!=0)
              {
                cout<<ps->syn<<" , ";
                ps=ps->next;
              }
            pw=pw->next;
          }
      }
}
```

المكدس - Stack

هو بنية خطية تشبه السلسلة المترابطة حيث يمكن معه إضافة عناصر و إزالتها من قمة المكدس فقط أي حسب المبدأ " الداخل أخيراً هو الخارج أولاً " (LIFO = Last in , First out) .
يمكن استخدام المكدسات ضمن العديد من التطبيقات الهامة. على سبيل المثال, يحتاج التابع المستدعى أن يعرف كيف يمكن له أن يعود إلى التابع الذي قام باستدعائه, لذلك يتم وضع عنوان مكان العودة ضمن مكدس. فإذا حدثت عدة استدعاءات لتوابع , يتم تخزين عناوين العودة الخاصة بها ضمن مكدس حسب المبدأ LIFO مما يساعد على عودة كل تابع إلى التابع الذي قام باستدعائه. تساعد المكدسات على التعامل مع التوابع العودية و استدعاءاتها بنفس الطريقة التي تتعامل فيها مع التوابع التقليدية غير العودية. أيضاً يتم استخدام المكدسات مع المترجمات من أجل تحديد طريقة التعامل مع التعابير لحساب قيمتها و توليد لغة الآلة الموافقة لها.

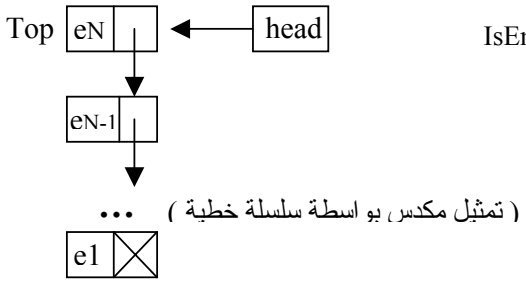
العمليات على المكدس:

EmptyStack : إنشاء مكدس فارغ (للتهيئة)
Push : إضافة عنصر إلى المكدس
Pop : حذف عنصر من المكدس
Top : قراءة العنصر الموجود في قمة المكدس
IsEmpty : اختبار كون المكدس فارغاً
CleanUp : تفريغ المكدس



ملاحظات: بفرض S مكدساً فإن :

- العمليات Pop(s) و Top(s) تكون معرفة إذا و فقط إذا كان IsEmpty(s) = false
- Pop(Push(s,e)) = s
- Top(Push(s,e)) = e
- IsEmpty(EmptyStack) = true
- IsEmpty(Push(s,e)) = false



يمكن تمثيل المكدس بطريقتين : باستعمال الأشعة و باستعمال المؤشرات

لنقوم بتمثيل مكدساً بالطريقتين و نضع كل تمثيل مع عملياته في ملف رأسي حتى نستطيع الاستفادة منهما في الأمثلة الآتية:

التمثيل باستعمال المؤشرات

```
//  save in header file : Stack_P.h
#include <iostream.h>
struct stackElem {
         int val;
         stackElem *link;
};
typedef stackElem* stack;
/*************************************/
void emptyStack(stack &s)
{
  s=0;       // nil
}
/*************************************/
char isEmpty(stack s)
{
  if (s==0)
     return 'y';
  else
     return 'n';
}
/*************************************/
```

التمثيل باستعمال الأشعة

```
//  save in header file : Stack_A.h
#include <iostream.h>
const lmax=100;
struct stack {
       int top;
       int val[lmax];
};
/*************************************/
void emptyStack(stack &s)
{
  s.top=-1;
}
/*************************************/
char isEmpty(stack s)
{
  if (s.top==-1)
      return 'y';
  else
      return 'n';
}
/*************************************/
```

```
void push(stack &s,int e)
{
  if (s.top==lmax-1)
      cout<<"Error: Stack is full \n";
  else
      {
        ++s.top;
        s.val[s.top]=e;
      }
 }
/*************************************/
void pop(stack &s)
{
  if (isEmpty(s)=='y')
      cout<<"Error: Stack is empty \n";
  else
      --s.top;
}
/*************************************/
int top(stack s)
{
  if (isEmpty(s)=='y')
      cout<<"Error: Stack is empty \n";
  else
      return s.val[s.top];
}
/*************************************/
void cleanUp(stack &s)
{
  if (isEmpty(s)=='y')
     cout<<"Stack is already cleaned up";
  else
      emptyStack(s);
}
```

```
void push(stack &s,int e)
{
  stack h=new stackElem();
  h->val=e;
  h->link=s;
  s=h;
}
/*************************************/
void pop(stack &s)
{
  stack h=s;
  if (isEmpty(s)=='y')
      cout<<"Error: Stack is empty \n";
  else
      {
        s=s->link;
        delete h;
      }
}
/*************************************/
int top(stack s)
{
  if (isEmpty(s)=='y')
      cout<<"Error , Stack is empty \n";
  else
      return s->val;
}
/*************************************/
void cleanUp(stack &s)
{
  if (isEmpty(s)=='y')
     cout<<"Stack is already cleaned up";
  else
      while (isEmpty(s)=='n')
        pop(s);
}
```

مثال عملي : أعد كتابة برنامج حساب العاملي لعدد صحيح باستخدام مفهوم المكدسات.

الحل: بالاعتماد على الوحدة Stack_P.h نكتب ما يلي :

```
#include <iostream.h>
#include "Stack_P.h"
int n;

long fact(int n);
/*************************************************************************************/
void main()
{
  cout<<"Enter n= ";
  do { cin>>n;
      } while (n<0);
  cout<<n<<"! = "<<fact(n)<<endl;
}
/*************************************************************************************/
long fact(int n)
{
  stack s;
  emptyStack(s);
  long f=1;
  while (n>0)
    { push(s,n);
      --n;
    }
```

```
  while (isEmpty(s)=='n')
    { f*=top(s);
      pop(s);
    }
  return f;
}
```

ملاحظة هامة: إن الفرق ما بين قائمة مترابطة و مكدس هو أنه يمكن إدخال و حذف عنصر في القائمة في أي مكان نريد , أما بالنسبة للمكدسات فيمكن إدراج العناصر و حذفها من قمة المكدس فقط.

تمرين :

اكتب إجرائية تقوم بطباعة عبارة رياضية بعد تحويلها من الشكل النظامي (Infix) إلى الشكل الملحق (postfix) مستخدماً مفهوم المكدسات .

الحل:

لنقوم بإنشاء ملف رأسي جديد "Stack_PC.h" , حيث يحوي هذا الملف جميع عمليات المكدس و لكن مع الفرق أن محتوى السجلات من نوع محرفي char بدل عدد صحيح int . و بالتالي نجد :

```
#include <iostream.h>
#include <string.h>
#include "Stack_PC.h"

char c[25];

void print(char s[]);
/*****************************************************************************/
void main()
{
  cout<<"Enter a mathimatical infix sentence : \n";
  cin>>c;
  print(c);
}
/*****************************************************************************/
void print(char c[])
{ stack s;
  emptyStack(s);
  for (int i=0;i<strlen(c);i++)
      {
        if ((int(c[i])>=48)&&(int(c[i]<=57)))  // Ascci : 0 - 9
            cout<<c[i];
        else
        if ((c[i]=='+')||(c[i]=='-')||(c[i]=='*')||(c[i]=='/')||(c[i]=='(')
                                                          ||(c[i]==')'))
            switch (c[i])
            {
              case '+':case '-':
                      { while ((isEmpty(s)=='n')&&(top(s)!='('))
                          { cout<<top(s);
                            pop(s);
                          }
                        push(s,c[i]);
                        break;
                      }
              case '*':case '/':
                      { if ((isEmpty(s)=='n')&&((top(s)=='*')||(top(s)=='/')))
                           { cout<<top(s);
                             pop(s);
                           }
                        push(s,c[i]);
                        break;
                      }
              case '(':{
                        push(s,c[i]);
```

```
                        break;
                       }
              case ')':{
                        while (top(s)!='(')
                          { cout<<top(s);
                            pop(s);
                          }
                        pop(s);   // pop '('
                        break;
                       }
            } // end of switch
      } // end of for
  while (isEmpty(s)=='n')
     { cout<<top(s);
       pop(s);
     }
}
```

❖ **تمرين :**

اكتب إجرائية لحساب قيمة تعبير رياضي مكتوب بالشكل النظامي (حيث يمكن للتعبير أن يحوي العمليات الثنائية البسيطة : + , - , * و / , بالإضافة إلى الأقواس "(" و ")" .

الحل:

لننشئ ملفين رأسيين لتمثيل نوعين من المكدسات : الأول تكون عناصره من نوع float من أجل تخزين الأعداد و الثاني عناصره char من أجل تخزين العمليات و الأقواس , حيث يجب مناقشة الأولويات بالشكل التالي :

عدد : push

- , + : نخرج جميع العمليات الأولى منها (بشرط أن المكدس ليس فارغاً و القمة ليست "(") , ثم push (حيث أن العمليات الأولى هي : * , / , + , - (+ , - : لأنها موجودة على اليسار))

/ , * : نخرج جميع العمليات الأولى منها (بشرط أن المكدس ليس فارغاً و القمة ليست "(") , ثم push (حيث أن العمليات الأولى هي : * , / لأنها موجودة على اليسار)

(: push

) : نخرج جميع العمليات إلى أن نصل إلى أقرب قوس مفتوح مقابل له أي "(" , ثم pop لـ "(" .

```
#include <iostream.h>
#include <string.h>      //  --> strlen()
#include "Stack_PF.h"    //  name of Stack= "stack"   ,  val = float
#include "Stack_PC.h"    //  name of Stack= "Cstack"  ,  val = char

char s[25];
float calculate(char s[]);
float getNumber(char s[],int &i);  //  تابع للحصول على عدد مؤلف من عدة أرقام متتالية
/*************************************************************************/
void main()
{
  cout<<"Enter a mathimatical infix sentence : \n";
  cin>>s;
  cout<<s<<" = "<<calculate(s);
}
/*************************************************************************/
float calculate(char s[])
{
  float n1,n2; char op;
  stack sF;        Cstack sC;
  emptyStack(sF);  emptyStack(sC);
  for (int i=0;i<strlen(s);i++)
      {
       if ((int(s[i])>=48)&&(int(s[i]<=57)))  // Ascci : 0 - 9
           push(sF,getNumber(s,i));
       else
```

```
      if ((s[i]=='+')||(s[i]=='-')||(s[i]=='*')||(s[i]=='/')||(s[i]=='(')||(s[i]==')'))
         switch (s[i]){
         case '*':
         case '/':{if ((isEmpty(sC)=='n')&&((top(sC)=='*')||(top(sC)=='/')))
                     {
                       n1=top(sF);  pop(sF);
                       n2=top(sF);  pop(sF);
                       op=top(sC);  pop(sC);
                       if (op=='*') push(sF,n1*n2);
                       else         push(sF,n2/n1);     //  op == '/'
                     }
                   push(sC,s[i]);
                   break;
                  }
         case '+':
         case '-':{while ((isEmpty(sC)=='n')&&(top(sC)!='('))
                     {
                       n1=top(sF);  pop(sF);
                       n2=top(sF);  pop(sF);
                       op=top(sC);  pop(sC);
                       if (op=='+') push(sF,n1+n2);
                       else
                       if (op=='-') push(sF,n2-n1);
                       else
                       if (op=='*') push(sF,n1*n2);
                       else         push(sF,n2/n1);     //  op == '/'
                     }
                   push(sC,s[i]);
                   break;
                  }
         case '(':{ push(sC,s[i]);  break; }

         case ')':{
                   while ((isEmpty(sC)=='n')&&(top(sC)!='('))
                     {
                       n1=top(sF); pop(sF);
                       n2=top(sF); pop(sF);
                       op=top(sC); pop(sC);
                       if (op=='+') push(sF,n1+n2);
                       else
                       if (op=='-') push(sF,n2-n1);
                       else
                       if (op=='*') push(sF,n1*n2);
                       else         push(sF,n2/n1);     //  op == '/'
                     }
                   pop(sC);   // pop '('
                   break;
                  }
         } // end of switch
    } // end of for
  while (isEmpty(sC)=='n')
    { n1=top(sF);  pop(sF);
      n2=top(sF);  pop(sF);
      op=top(sC);  pop(sC);
      if (op=='+') push(sF,n1+n2);
      else
      if (op=='-') push(sF,n2-n1);
      else
      if (op=='*') push(sF,n1*n2);
      else         push(sF,n2/n1);     //  op == '/'
    }
  return top(sF);
}
```

```
/*************************************************************************************/

float getNumber(char s[],int &i)
{
  float num=0;
  while ((int(s[i])>=48)&&(int(s[i])<=57))
    {
      num=num*10+int(s[i])-int('0');
      ++i;
    }
  --i;
  return num;
}
```

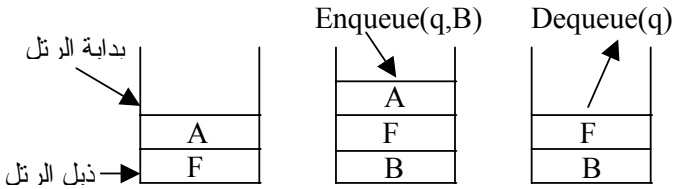
الرتل (صف الانتظار) - Queue

و هو بنية خطية تشبه القائمة المترابطة غير أن عمليات الإضافة تجري في جهة ندعوها ذيل الرتل tail , و يجري الحذف في الجهة المعاكسة التي نسميها بداية الرتل head, و بالتالي فإن الأرتال تعمل وفقاً للمبدأ "الداخل أولاً هو الخارج أولاً" أي (FIFO = First in , First Out) .

يمكن استخدام الأرتال ضمن العديد من التطبيقات الهامة. فمثلاً من المعلوم أن معظم الحواسب لها معالج وحيد فقط , لذلك يمكنها القيام بخدمة مستثمر واحد فقط في لحظة معينة. يتم تخزين المعلومات المتعلقة بالمستثمرين الآخرين ضمن رتل . كما يمكن استخدام الأرتال لإدارة عمليات الطباعة في حال وجود عدة مستثمرين و طابعة واحدة فقط...

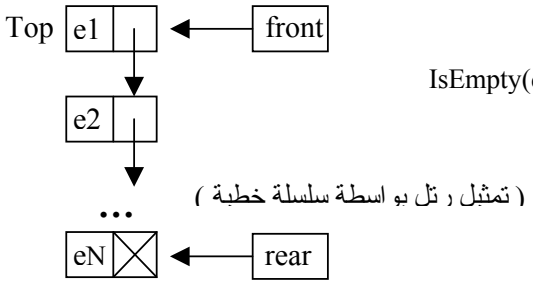
العمليات على الرتل:

EmptyQueue : إنشاء رتل فارغ (للتهيئة)
Enqueue : إضافة عنصر إلى الرتل
Dequeue : حذف عنصر من الرتل
Top : قراءة العنصر الموجود في بداية الرتل
IsEmpty : اختبار كون الرتل فارغاً
CleanUp : تفريغ الرتل



ملاحظات: بفرض أن q رتلاً , فإن :

- العمليات Dequeue(q) و Top(q) تكون معرفة إذا و فقط إذا كان IsEmpty(q) = false
- IsEmpty(EmptyQueue) = true
- IsEmpty(Enqueue(q,e)) = false



(تمثيل رتل بواسطة سلسلة خطية)

يمكن تمثيل الرتل بطريقتين: باستعمال الأشعة و باستعمال المؤشرات

يمثل الرتل في سلسلة خطية بحيث يؤشر إلى بدايتها مؤشر بداية الرتل front و يؤشر إلى نهايتها مؤشر ذيل الرتل tail .

لنقوم بتمثيل رتلاً بطريقة المؤشرات حيث نضع التمثيل مع عملياته ضمن ملف رأسي Queue_P.h حتى نستطيع الاستفادة منه في مثال برنامج حساب الـ Fibonacci :

برنامج Fibonacci

```
#include <iostream.h>
#include "Queue_P.h"

int n;
int fib(int n);
/*************************************/
void main()
{
  cout<<"Enter n = ";
  cin>>n;
  cout<<"Fib("<<n<<")= "<<fib(n);
}
/*************************************/
int fib(int n)
{
  int f1,f2;
  queue q;
  emptyQueue(q);
  if (n<=1) return n;
  else
     {
       enqueue(q,0);
```

تمثيل الرتل باستعمال المؤشرات

```
//  save in header file : Queue_P.h
#include <iostream.h>
struct elem {
       int   val;
       elem* link;
};
struct queue {
       elem* front;
       elem* tail;
};
/*************************************/
void emptyQueue(queue &q)
{
  q.front=0;
  q.tail=0;
}
/*************************************/
char isEmpty(queue q)
{ if (q.front==0)
       return 'y';
  else return 'n';
}
```

```
/************************************/
void enqueue(queue &q,int x)
{
  elem* h=new elem();
  h->val=x;
  h->link=0;
  if (isEmpty(q)=='y')
     {
       q.front=h;
       q.tail=h;
     }
  else
     {
       q.tail->link=h;
       q.tail=h;
     }
}
/************************************/
void dequeue(queue &q)
{
  if (isEmpty(q)=='y')
      cout<<"Error: Queue is empty \n";
  else
      {
        elem* h=q.front;
        if (q.front->link==0)
           {
             q.front=0;
             q.tail=0;
           }
        else
             q.front=q.front->link;
        delete h;
      }
}
/************************************/
int top(queue q)
{
  if (isEmpty(q)=='y')
      cout<<"Error: Queue is empty \n";
  else
      return q.front->val;
}
/************************************/
void cleanUp(queue &q)
{
  if (isEmpty(q)=='y')
      cout<<"Error: Queue is already
                          cleaned up \n";
  else
      while (isEmpty(q)=='n')
         dequeue(q);
}
```

```
       enqueue(q,1);
       for (int i=2;i<=n+1;i++)
           {
             f1=top(q);
             dequeue(q);
             f2=top(q);
             enqueue(q,f1+f2);
           }
       return top(q);
     }
}
```

برنامج إيجاد القاسم المشترك الأكبر لعددين GCD

```
#include <iostream.h>
#include "Queue_P.h"

int x,y;

int GCD(int x,int y);
/************************************/
void main()
{
  cout<<"Enter x,y : ";
  cin>>x>>y;
  cout<<"GCD of ("<<x<<","<<y<<") =
                                "<<GCD(x,y);
}
/************************************/
int GCD(int x,int y)
{
  int f1,f2;
  queue q;
  emptyQueue(q);
  enqueue(q,y);
  enqueue(q,x);
  while (top(q)!=0)
    {
      f1=top(q);    dequeue(q);
      f2=top(q)%f1; dequeue(q);
      enqueue(q,f2);
      enqueue(q,f1);
    }
  dequeue(q);
  return top(q);
}
```

تمرين:
يصطف عدد من المواطنين أمام فرن لبيع الخبز و الفطائر , بعد دفع الحساب يتم فصل الزبائن إلى طابورين , الأول لبيع الخبز و الثاني لبيع الفطائر. و يتم البيع حسب الدور , علماً أنه يجب معرفة نوع الطلب (خبز/فطائر) و الكمية (عدد الأرغفة/عدد الفطائر) لكل زبون. و المطلوب:

1- اكتب تمثيلاً تعبر فيه عما سبق (باستخدام المؤشرات) .
2- اكتب خوارزمية تقوم بفصل الطابور الرئيسي إلى طابورين حسب نوع الطلب .
3- اكتب خوارزمية تقوم بحساب الكمية التي باعها صاحب الفرن من الخبز أو الفطائر .

الحل:

```
struct node{
       char  kind;    // '0'= bread , '1'= pie
       int   num;
       node* next;
};
struct Queue{
       node *top,*tail;
};
/*************************************************************************************/
void sort(Queue q,Queue &qB,Queue &qP)
{ Queue h;
  while (isEmpty(q)=='n')
    {
      h=top(q);
      if (h->kind=='0')     //  = bread
          enqueue(qB,h);
      else
          enqueue(qP,h);    //   = pie
      dequeue(q);
    }
}
/*************************************************************************************/
void amount(Queue q,int &am)
{ Queue h;
  while (isEmpty(q)=='n')
    {
      h=top(q);
      am+=h->num;
      dequeue(q);
    }
}
```

تمرين:
اكتب خوارزمية لطباعة شجرة ثنائية على شكل مستويات (أي طباعة العقد بدءاً من الجذر ثم تمشيط عقد المستوي الأول ثم الثاني ... حتى المستوي الأخير) .

الحل:

```
void print(BTree b)
{ Queue q; emptyQueue(q);
  if (b!=0)
     {
       enqueue(q,b);
       while (isEmpty(q)=='n')
         {
           b=Top(q); dequeue(q);
           cout<<b->val;
           if (b->l!=0) enqueue(q,b->l);
           if (b->r!=0) enqueue(q,b->r);
         }
     }
}
```

تمرين:

اكتب خوارزمية للتحقق من كون شجرة بحث ثنائي هي كاملة أم لا (isPerfect) .

```
char isPerfect(BTree b)
{
  Queue q;  emptyQueue(q);
  char c='n';
  if (b!=0) enqueue(q,b);
  while (isEmpty(q)=='n')
    {
      b=Top(q); dequeue(q);
      if (b==0) c='y';
      else
          if (c=='y') return 'n';
      if (b!=0)
         { enqueue(q,b->l);
           enqueue(q,b->r);
         }
    }
  return 'y';
}
```

تحويل الخوارزميات العودية إلى خوارزميات تكرارية

2- الطريقة العامة لحذف الاستدعاء العودي غير المتطرف:

نقول عن استدعاء عودي أنه غير متطرفاً إذا كان يوجد بعد هذا الاستدعاء تعليمة واحدة على الأقل .
إن الشكل العام للتحويل كالتالي (حيث أن الاستدعاء العودي P( ) ليس متطرفاً , A و B و C كتل من التعليمات و حيث أن هنا لا توجد وسطاء تمرير للإجراء) :
طالما لا توجد وسطاء تمرير (= معطيات جديدة) فإذاً يتم في كل استدعاء عودي إعادة تنفيذ التعليمات نفسها تماماً دون أي فرق , لذلك يمكن تحديد label في نقطة معينة تناسب المسألة المطروحة , و لكن بما أننا نذهب في كل مرة إلى هذا الـ label مقابل كل استدعاء عودي فمن المستحيل متابعة تنفيذ التعليمات الموجودة ما بعد الاستدعاء العودي (B) لأن تعليمة goto تقطع لنا في كل مرة الطريق الواصل إلى تلك التعليمات (B) , لذلك يجب إضافة label آخر يعطي مكان هذه التعليمات المقطوعة. من الواضح أنه يجب تنفيذ (B) لأول مرة بعد أن نصل إلى شرط التوقف (أي بعد تنفيذ (C) التي لا تنفذ إلا مرة واحدة فقط !!!) . عدد مرات تنفيذ (B) يتوقف على عدد الاستدعاءات العودية , لذلك قمنا بإضافة عداد في بداية الخوارزمية يقوم بعملية العد و بالتالي فإن (B) تنفذ بقيمة العداد مرة .

الخوارزمية بشكل عودي	حذف الاستدعاء العودي	طريقة ثانية للحذف

`void P()`
`{`
`  if (condition)`
`  {`
`  A ;`
`  P();`
`  B ;`
`  }`
`  else`
`  C ;`
`}`
	`Void P()`
	`{ int i=0;`
	`Start:`
	`  if (condition)`
	`  {`
	`  A ;`
	`  ++i;`
	`  goto Start;`
	`L1:  B ;`
	`  --i;`
	`  }`
	`  else`
	`  C ;`
	`  if (i>0) goto L1;`
	`}`
		`void P()`
		`{ int i=0;`
		`  while (condition)`
		`  {`
		`  A ;`
		`  ++i;`
		`  }`
		`  C;`
		`  while (i>0)`
		`  {`
		`  B ;`
		`  --i;`
		`  }`
		`}`

ملاحظة : في حال وجود عدة استدعاءات عودية يمكن استخدام عدة labels و توزيعها بالشكل المناسب , بالإضافة إلى مكدساً نضع فيه أرقاماً تدل على الاستدعاءات العودية , و نستعمل هذه القيم في تحديد آخر استدعاء مؤجل .
ملاحظة : عندما توجد وسطاء تمرير للتابع العودي فإن في كل استدعاء عودي يتم تنفيذ التعليمات نفسها و لكن من أجل معطيات مختلفة , فإذا كان لدينا تعليمات ما بعد الاستدعاء العودي متعلقة بهذه المعطيات , يجب علينا حفظ هذه المعطيات قبل تغييرها من أجل إمكانية استرجاعها أثناء طريق العودة من استدعاء ابن إلى استدعاء أب . إذاً هنا لا يكفينا عداد يقوم بعد عدد مرات الاستدعاءات العودية , و إنما يمكن استخدام مفهوم المكدسات لتخزين المعطيات مؤقتاً فيها . إن المثال التالي يبين هذه الفكرة :

❖ **مثال :** اكتب خوارزمية غير عودية لمسألة أبراج هانوي (معتمداً مفهوم المكدسات) .

الحل :

```
#include <iostream.h>
#include "Stack_P.h"    //  name of Stack = stack  ,  val = int
#include "Stack_PC.h"   //  name of Stack = Cstack  ,  val = char
/*************************************************************************/
void Hanoi(int n,char a,char b,char c)
{ stack sN;  Cstack sA,sB,sC;  char aux;
  emptyStack(sN); emptyStack(sA); emptyStack(sB); emptyStack(sC);
start:
  if (n==1) cout<<a<<" --> "<<c<<endl;
  else
     {
       push(sN,n);
       push(sA,a);
       push(sB,b);
       push(sC,c);
       --n;
```

```
        aux=b; b=c; c=aux;
        goto start;
L:
        cout<<a<<" --> "<<c<<endl;
        --n;
        aux=a; a=b; b=aux;
        goto start;
      }
  if (isEmpty(sN)=='n')
     {
       n=top(sN); pop(sN);
       a=top(sA); pop(sA);
       b=top(sB); pop(sB);
       c=top(sC); pop(sC);
       goto L;
     }
}
```

تمرين:

اكتب خوارزمية تكرارية لطباعة شجرة ثنائية بالترتيب الملحق PostOrder .

الحل:

```
void printPostOrder(BTree b)
{
  Stack s1,s2,s3; emptyStack(s1); emptyStack(s2); emptyStack(s3);
start:
  if (b!=0)
     {
       push(s1,b); push(s3,b);
       b=b->L;
       goto start;
L1:    push(s2,b);
       b=b->R;
       goto start;
L2:    cout<<b->val;
     }
  if ((isEmpty(s2)=='n')&&(top(s2)==(top(s3))
     {
       b=top(s2); pop(s2); pop(s3);
       goto L2;
     }
  if (isEmpty(s1)=='n')
     {
       b=top(s1); pop(s1);
       goto L1;
     }
}
```


البنى الشجرية – Tree Structures


مقدمة :

إن الأشجار هي نوع من أنواع بنى المعطيات غير التسلسلية أي غير خطية (Non Sequential Structures) , والتي يمكن تعريفها كما يلي :

هي البنى التي لا توجد علاقة ترتيب كلي بين عناصرها (قد توجد علاقة ترتيب جزئي بين عناصر بعض البنى غير التسلسلية).

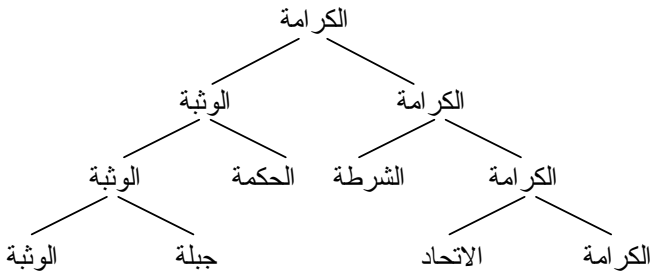
- تعريف الشجرة (Tree) :

هي مجموعة من العناصر تسميها *عقداً* nodes , ومرتبطة بعضها ببعض *بروابط* links , و منظمة تنظيماً هرمياً (Hierarchical) لا يحوي حلقات cycles مغلقة , أي أنه توجد عقدة مميزة و وحيدة تسميها *//جذر* root , يمكن بواسطتها الوصول إلى مجموعة من العقد , و من هذه العقد إلى عقد أخرى و هكذا ...

تعتبر البنى الشجرية من البنى المميزة في مجال المعلوماتية , نظراً لكثرة استعمالها في نظم الاستثمار خاصة , حيث تنظم الملفات في مجلدات يحوي كل منها مجموعة من المجلدات الجزئية و مجموعة من الملفات .

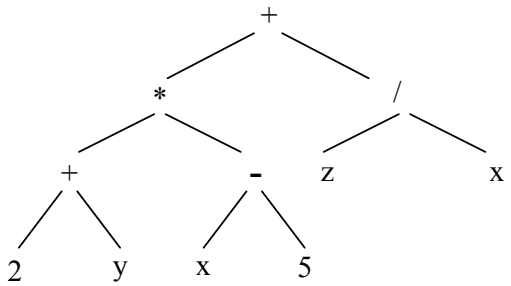
- أمثلة على الأشجار :

تمثيل نتيجة دوري كرة القدم باستخدام شجرة ثنائية :



تمثيل عبارة حسابية باستخدام شجرة ثنائية :

((2+y)*(x-5))+(z/x)



من أهم أنواع الأشجار لدينا الأشجار الثنائية Binary Trees و الأشجار المعممة Generalized Trees .

❖ الأشجار الثنائية - Binary Trees :

و هي الأشجار التي يكون فيها لكل عقدة ابنان على الأكثر .

❑ تعاريف :

تعريف 1 : نسمي شجرة ثنائية كل بنية B تحقق أحد الشرطين التاليين :

- $B = \Phi$ (الشجرة فارغة)
- $B = < O, B1, B2 >$ حيث O عقدة نسميها الجذر . B1 , B2 شجرتان ثنائيتان منفصلتان . نسمي B1 الشجرة الجزئية اليسارية و B2 الشجرة الجزئية اليمينية .

تعريف 2 :

نسمي جذر الشجرة الجزئية B1 الابن الأيسر لـ O , و جذر الشجرة الجزئية B2 الابن الأيمن لـ O , و نقول إنه يوجد اتصال يساري بين الجذر و الابن الأيسر و اتصال يميني بين الجذر و الابن اليميني .

تعريف 3 :

يمكن أن توجد بين العقد العلاقات التالية : أب , ابن , أخ , سلف و حفيد . نقول عن عقدتين أنهما أختان , إذا كان لهما الأب نفسه.

نقول عن عقدة n_i أنها سلف (Ancestor) لعقدة n_j , إذا و فقط إذا كانت n_i أباً لـ n_j , أو كانت n_i سلفاً لأب n_j .

نقول عن عقدة n_i أنها حفيد (Grandchild) لعقدة n_j , إذا و فقط إذا كانت n_i ابناً لـ n_j , أو كانت n_i حفيداً لابن n_j .

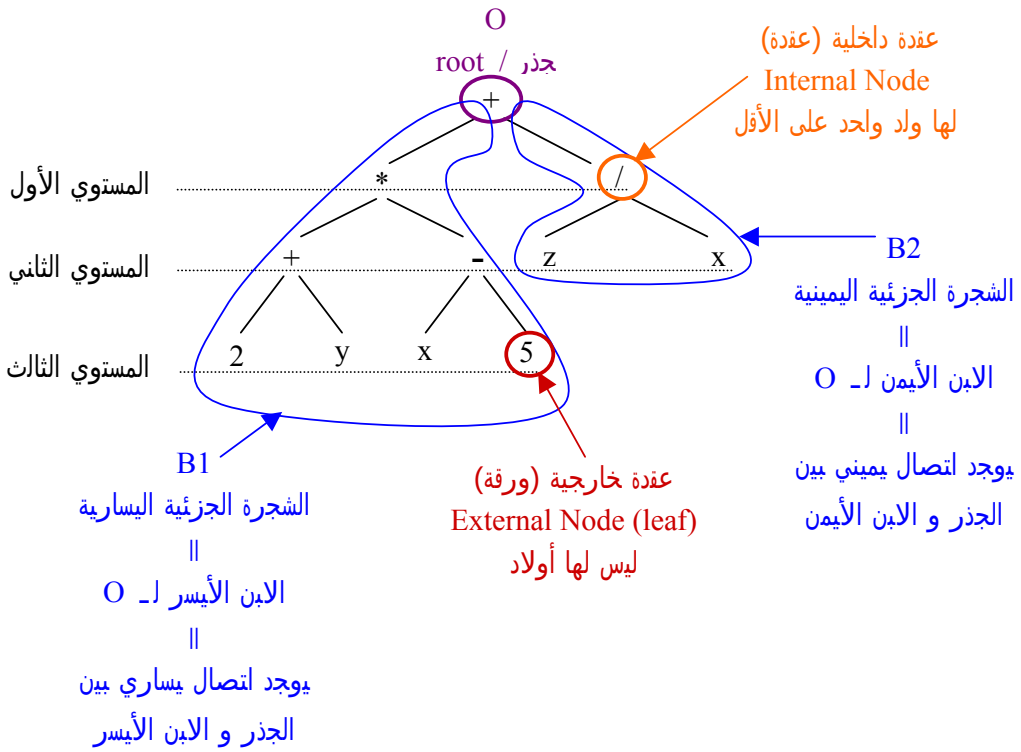
تعريف 4 :

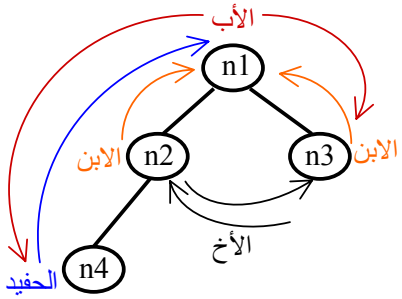
لكل عقدة في شجرة ثنائية ولدان على الأكثر :

- نسمي العقد التي لها ولد على الأقل عقداً داخلية (Internal Nodes) , أو اختصاراً عقداً .
- نسمي العقد التي ليس لها أولاد عقداً خارجية (External Nodes) , أو أوراق (Leafs) .

تعريف 5 :

نسمي طريقاً (Path) ضمن شجرة ثنائية كل متتالية من العقد , و نسمي فرعاً في الشجرة كل طريق يصل بين الجذر و إحدى الأوراق .



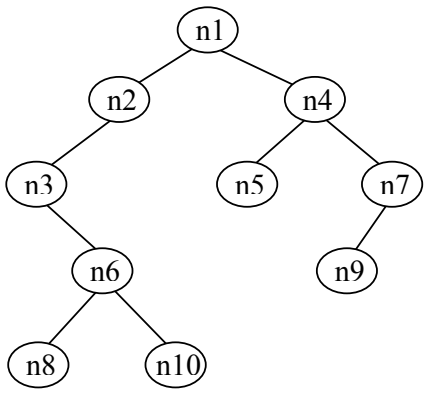


❑ بعض القياسات على الأشجار الثنائية :

بفرض أن B شجرة ثنائية و x عقدة في هذه الشجرة :

1	درجة الشجرة D(B)	العدد الأعظم لأولاد عقدة ← درجة شجرة ثنائية D(B) = 2
2	حجم الشجرة Vol(B)	عدد عقدها ← Vol(empty tree) = 0 Vol(< O,B1,B2 >) = 1 + Vol(B1) + Vol(B2)
3	ارتفاع عقدة h(x)	عدد الاتصالات في الطريق الواصل بين هذه العقدة و الجذر ← h(x) = 0 ; x = root h(x) = 1 + h(y) ; y = father of x
4	ارتفاع الشجرة H(B)	أطول ارتفاع عقدة في الشجرة : H(B) = max{ h(x) ; x is a node of B }
5	عرض الشجرة L(B)	العدد الأعظم للعقد في مستوي ما
6	مسافة التجول ضمن شجرة LC(B)	مجموع ارتفاعات عقدها : LC(B) = Σ h(x) ; x is a node of B
7	مسافة التجول الداخلي ضمن شجرة LCI(B)	مجموع ارتفاعات عقدها الداخلية : LCI(B) = Σ h(x) ; x = internal node of B
8	مسافة التجول الخارجي ضمن شجرة LCE(B)	مجموع ارتفاعات أوراقها : LCE(B) = Σ h(x) ; x = external node of B

مثال : لتكن B الشجرة الثنائية المبينة في الشكل التالي , فنلاحظ أن :

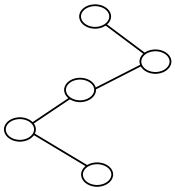


Root(B) = n1
Left Son (n1) = n2
Right Son (n1) = n4
Leafs = { n8 , n10 , n5 , n9 }
Internal Nodes = { n1 , n2 , n3 , n4 , n6 , n7 }
h(n1) = 0 , h(n2) = h(n4) = 1 , h(n3) = h(n5) = h(n7) = 2
h(n6) = h(n9) = 3 , h(n8) = h(n10) = 4
H(B) = 4
L(B) = 3
LC(B) = 22
LCI(B) = 9
LCE(B) = 13

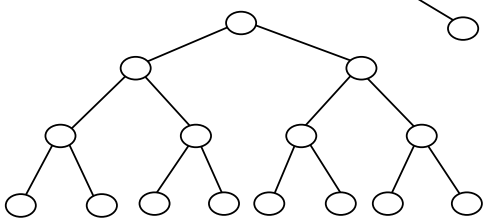
ملاحظة : نلاحظ أن LC(B) = LCI(B) + LCE(B)

❑ بعض الأشجار الثنائية الخاصة :

1- الشجرة الثنائية الخطية (Linear) : لكل عقدة ولد واحد على الأكثر .



2- الشجرة الثنائية التامة (Complete) : في المستوي 0 ← عقدة واحدة
في المستوي 1 ← عقدتان
في المستوي 2 ← 4 عقد
في المستوي k ← 2^k عقدة

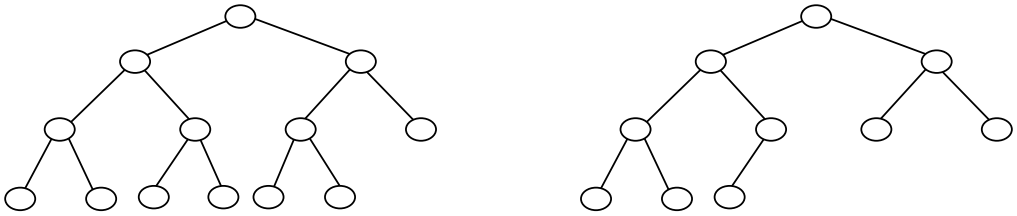


← عدد العقد في الشجرة B التي ارتفاعها h : $Vol(B) = 1 + 2 + 4 + ... + 2^h = 2^{h+1} - 1$

أي أن جميع المستويات مملوءة تماماً بالعقد .

3- الشجرة الثنائية الكاملة (Perfect) :

جميع المستويات مملوءة تماماً بالعقد , ما عدى المستوي الأخير الذي يمكن أن يحوي فراغات ,
و في هذه الحالة تكون الفراغات دوماً في جهة اليمين .

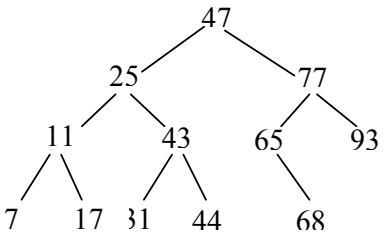


4 - أشجار البحث الثنائي Binary Search Trees :

هي أشجار لا تتضمن قيماً مكررة حيث أن العقد الموجودة على الطرف الأيسر لأية شجرة جزئية هي عقد ذات قيم أصغر من قيمة العقدة الأب لها , و العقد الموجودة على الطرف الأيمن لأية شجرة جزئية تكون قيمها أكبر من قيمة العقدة الأب لها (أو بالعكس : توضع القيم الصغيرة على الطرف الأيمن و القيم الكبيرة على الطرف الأيسر) .

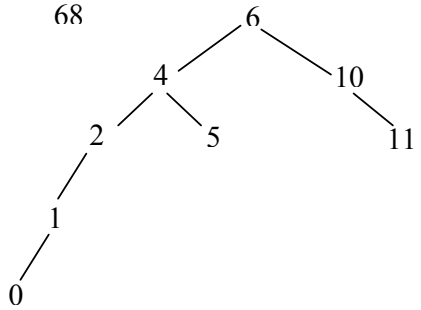
مثال 1 : لنشكل من القيم التالية شجرة بحث ثنائي :

{ 47 , 25 , 77 , 11 , 43 , 93 , 65 , 31 , 17 , 7 , 44 , 68 }



مثال 2 : لنشكل من القيم التالية شجرة بحث ثنائي :

{ 6 , 4 , 5 , 10 , 2 , 1 , 0 , 11 }



❑ بعض النظريات الخاصة بالأشجار الثنائية :

نظرية 1 : إذا كانت B شجرة ثنائية حجمها n و ارتفاعها H , عندئذ يكون : $\lfloor Log_2 n \rfloor \le H \le n-1$

حيث هو أكبر عدد صحيح أصغر من x .

نتيجة : كل شجرة ثنائية B غير فارغة ذات f ورقة , يكون ارتفاعها H أكبر أو يساوي $\lfloor Log_2 f \rfloor$

ملاحظة : في كل شجرة ثنائية حجمها n و تحوي f ورقة يكون : $f \le \frac{n+1}{2}$

نظرية 2 : لتكن B شجرة ثنائية حجمها n . يمكن حصر مسافة التجول في B بالمتراجحة : $\sum_{k=1}^{n} \lfloor Log_2 k \rfloor \le LC(B) \le \frac{n(n-1)}{2}$

نتيجة : مسافة التجول في شجرة ثنائية حجمها n , تكون في أحسن الأحوال من مرتبة $n \cdot Log_2 n$ و في أسوأ الأحوال n^2 .

نظرية 3 : لتكن B شجرة ثنائية تحوي f ورقة , يكون الارتفاع الوسطي للأوراق : $PF(B) \ge \log_2 f$

❑ تمثيل الأشجار الثنائية :

يمكن تمثيل الأشجار بطريقتين , إما باستخدام المصفوفات أو باستخدام المؤشرات .

1- تمثيل الأشجار الثنائية باستعمال المؤشرات : كل عقدة تُمثل بسجل من الشكل :

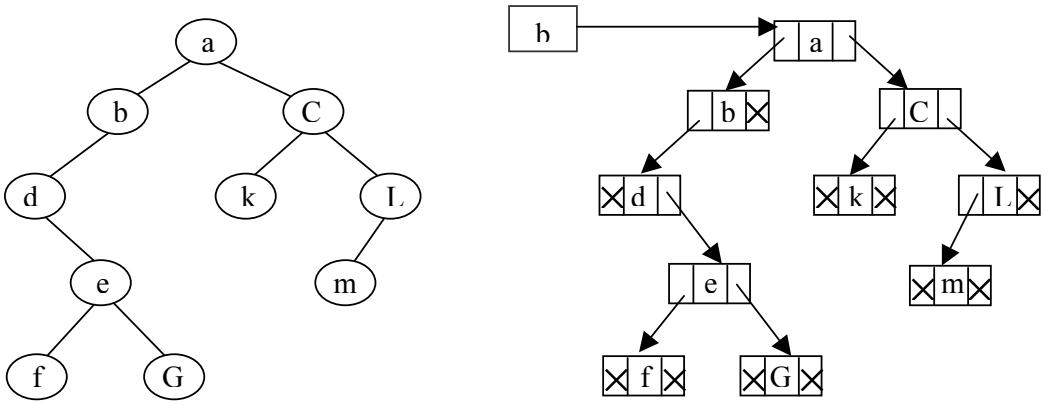
```
struct node {
       element val;    // int , char , ...
       node *L,*R;
};

typedef node* BTree;
BTree b;
```

حيث L مؤشر إلى الشجرة الجزئية اليسارية

R مؤشر إلى الشجرة الجزئية اليمينية

و بالتالي فإن الشجرة b هي عبارة عن مؤشر إلى العقدة الأولى (أي الجذر) .

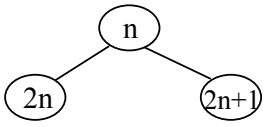
مثال:



2- تمثيل الأشجار الثنائية باستعمال المصفوفات :

نعرف مصفوفة فيها ثلاثة أعمدة بحيث يحوي كل سطر فيها معلومات العقدة , رقم الابن اليساري و رقم الابن اليميني , حيث يتم ترقيم العقد كما يلي :

ترقيم العقد ضمن شجرة ثنائية : إذا كان n رقم العقدة , فإن 2n هو رقم الابن اليساري , و 2n+1 هو رقم الابن اليميني . و بالتالي نستنتج أن أبا عقدة رقمها m هو العقدة رقم m/2 .




```
const n=100;
struct node {
       element val;      // int , char , float , ...
       int L,R;
};
typedef node BTable[n];
struct TabBTree {
       int root;
       BTable tab;
};
TabBTree b;
```

مثال : لنمثل نفس الشجرة السابقة باستخدام المصفوفات :

root
1

1	a	2	3
2	b	4	-
3	C	6	7
4	d	-	9
5	-	-	-
6	k	-	-
7	L	14	-
8	-	-	-
9	e	18	19
-	-	-	-
14	m	-	-
-	-	-	-
18	f	-	-
19	G	-	-

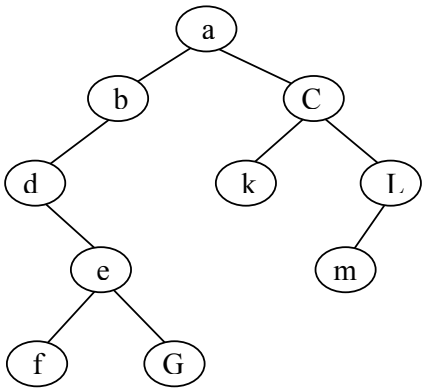
❑ التجول ضمن شجرة الثنائية - Traversal :

تهدف عملية التجول إلى المرور بكل عقدة من عقد الشجرة حسب ترتيب معين بهدف إجراء معالجة معينة عليها .

توجد ثلاثة طرق للتجول ضمن شجرة ثنائية , و هي كالتالي :

- التجول حسب الترتيب المصدر Preorder : يأتي الجذر أولاً , ثم الفرع اليساري و ثم الفرع اليميني
- التجول حسب الترتيب النظامي Inorder : يأتي الفرع اليساري أولاً , ثم الجذر و ثم الفرع اليميني
- التجول حسب الترتيب الملحق Postorder : يأتي الفرع اليساري أولاً , ثم الفرع اليميني و ثم الجذر

مثال: لتكن لدينا الشجرة الثنائية التالية و لنكتب محتويات هذه الشجرة بالاطرق الثلاثة :



Preorder : { a , b , d , e , f , G , C , k , L , m }
Inorder : { d , f , e , G , b , a , k , C , m , L }
Postorder : { f , G , e , d , b , k , m , L , C , a }

تمرين :

اكتب خوارزمية لكل مما يلي : 1) حساب عدد العقد الداخلية في شجرة بحث ثنائي 2) حساب عدد الأوراق في شجرة بحث ثنائي 3) حساب ارتفاع شجرة بحث ثنائي 4) حساب عرض شجرة بحث ثنائي 5) حساب مسافة التجول في شجرة بحث ثنائي 6) حساب مسافة التجول الداخلي في شجرة بحث ثنائي 7) حساب مسافة التجول الخارجي في شجرة بحث ثنائي 8) طباعة عقد شجرة بحث ثنائي بالترتيب النظامي 9) طباعة عقد شجرة بحث ثنائي بالترتيب المصدر 10) طباعة عقد شجرة بحث ثنائي بالترتيب الملحق 11) طباعة عقد شجرة بحث ثنائي الداخلية فقط 12) طباعة أوراق شجرة بحث ثنائي فقط 13) طباعة طريق عقدة في شجرة بحث ثنائي 14) التحقق من كون شجرة بحث ثنائي هي خطية 15) التحقق من كون شجرة بحث ثنائي هي تامة 16) التحقق من كون شجرة بحث ثنائي هي كاملة

الحل:

```
#include <iostream.h>
#include <conio.h>     // -->  clrscr() , getch() , getche()
#include <ctype.h>    //  -->  toupper()
#include <stdlib.h>  //   -->  _exit(0) (= exits the program and closes the window)
#include <math.h>   //    -->  pow()
#include "Queue_pBTree.h"   // for is_perfect
const m=100;
struct node{
       int val;
       node *l,*r;
};
typedef node* BTree;
BTree b=0,q=0;
int  n,i,maxH;
int a[m];  // for width
char c;   //  = choice of user

char menu();
void insertNode(BTree &b,int n);
void print(BTree b);        // choices of print
void printInfix(BTree b);
void printPrefix(BTree b);
void printPostfix(BTree b);
void printInternal(BTree b);
void printLeaf(BTree b);
void printPath(BTree b,int n);
int internalNum(BTree b);
int leafNum(BTree b);
void treeHeight(BTree b,int h,int &maxH);
void fillLevels(BTree b,int a[],int i);    //  for width
int width(BTree b);
int nodeHeight(BTree b,int x);
int LC(BTree b,BTree k);
int LCI(BTree b,BTree k);
int LCE(BTree b,BTree k);
char isLinear(BTree b);
char isComplete(BTree b);
char isPerfect(BTree b);
/*****************************************************************************/
void main()
{
  do{ clrscr();
      c=menu();
      switch (c){
      case '1':{cout<<"\n\nEnter the value to insert , val = ";
                cin>>n;
                insertNode(b,n);
```

```
                break;
               }
      case '2':{print(b);
                break;
               }
      case '3':{if ((b==0)||((b->l==0)&&(b->r==0)))
                     cout<<"\n\nThere are no internal nodes ! \n";
                else printInternal(b);
                break;
               }
      case '4':{if (b==0) cout<<"\n\nThere are no external nodes ! \n";
                else      printLeaf(b);
                break;
               }
      case '5':{cout<<"\nEnter the node you want to see its path , n = ";
                cin>>n;
                printPath(b,n);
                break;
               }
      case '6':{cout<<"\nThis tree contains "<<internalNum(b)<<" internal nodes\n";
                break;
               }
      case '7':{n=0;
                cout<<"\nThis tree contains "<<leafNum(b)<<" leaves !\n";
                break;
               }
      case '8':{maxH=0;
                treeHeight(b,0,maxH);
                cout<<"\nThe height of this tree is H(B)= "<<maxH<<endl;
                break;
               }
      case '9':{cout<<"\nThe width of this tree = "<<width(b);
                break;
               }
      case 'M':{cout<<"\nLC(B) = "<<LC(b,b);
                break;
               }
      case 'I':{cout<<"\nLCI(B) = "<<LCI(b,b);
                break;
               }
      case 'E':{cout<<"\nLCE(B) = "<<LCE(b,b);
                break;
               }
      case 'L':{if (isLinear(b)=='y') cout<<"\nYes , this tree is linear !\n";
                else                  cout<<"\nNo , this tree isn't linear !\n";
                break;
               }
      case 'C':{if (isComplete(b)=='y') cout<<"\nYes , this tree is complete !\n";
                else                    cout<<"\nNo , this tree isn't complete!\n";
                break;
               }
      case 'P':{
                if (isPerfect(b)=='y') cout<<"\nYes , tree is perfect !\n";
                else                   cout<<"\nNo , this tree isn't perfect !\n";
                break;
               }

      case '0': _exit(0);
      } // end of switch
      getch();
  } while (c!=0);
}
/*****************************************************************************/
```

```
char menu()
{ char ch;
  do{
      clrscr();
      cout<<"0: Exit \n";
      cout<<"1: Insert Node \n";
      cout<<"2: Print Tree \n";
      cout<<"3: Print Internal Nodes \n";
      cout<<"4: Print External Nodes (Leafs)\n";
      cout<<"5: Print Path Of A Node \n";
      cout<<"6: Number Of Internal Nodes \n";
      cout<<"7: Number Of External Nodes \n";
      cout<<"8: Height Of Tree \n";
      cout<<"9: Width Of Tree \n";
      cout<<"M: LC Of Tree \n";
      cout<<"I: LCI Of Tree \n";
      cout<<"E: LCE Of Tree \n";
      cout<<"L: Is The Tree Linear ? \n";
      cout<<"C: Is The Tree Complete ? \n";
      cout<<"P: Is The Tree Perfect ? \n\n";
      cout<<"Enter your choice : ";
      ch=getche(); cout<<endl;
      ch=toupper(ch);
  } while (((ch!='M')&&(ch!='I')&&(ch!='E')&&(ch!='L')&&(ch!='C')&&(ch!='P'))&&
                                                    ((int(ch)<48)||(int(ch)>57)));
  return ch;
}
/*****************************************************************************************/
void insertNode(BTree &b,int n)
{
  if (b==0)
      { b=new node; b->val=n; b->l=0; b->r=0; }
  else
  if (n<b->val) insertNode(b->l,n);
  else          insertNode(b->r,n);
}
/*****************************************************************************************/
void print(BTree b)
{ char p;
  do{
      clrscr();
      cout<<"1: Print Infix-Traversal \n";
      cout<<"2: Print Prefix-Traversal \n";
      cout<<"3: Print Postfix-Traversal \n\n\n";
      cout<<"Enter your choice : ";
      p=getche(); cout<<endl;
  } while ((p!='1')&&(p!='2')&&(p!='3'));
  if (b==0) cout<<"\n\nThis tree is empty ... , press any key to continue ... \n";
  else
  if (p=='1') printInfix(b);
  else
  if (p=='2') printPrefix(b);
  else        printPostfix(b);
}
/*****************************************************************************************/
void printInfix(BTree b)
{ if (b!=0)
     {
       printInfix(b->l);
       cout<<b->val<<"  ";
       printInfix(b->r);
     }
}
```

```
/*****************************************************************************/
void printPrefix(BTree b)
{
  if (b!=0)
     {
       cout<<b->val<<"  ";
       printPrefix(b->l);
       printPrefix(b->r);
     }
}
/*****************************************************************************/
void printPostfix(BTree b)
{
  if (b!=0)
     {
       printPostfix(b->l);
       printPostfix(b->r);
       cout<<b->val<<"  ";
     }
}
/*****************************************************************************/
void printInternal(BTree b)
{
  if ((b!=0)&&((b->l!=0)||(b->r!=0)))
     {
       cout<<b->val<<"  ";
       if (b->l!=0) printInternal(b->l);
       if (b->r!=0) printInternal(b->r);
     }
}
/*****************************************************************************/
void printLeaf(BTree b)
{ if (b!=0)
     { if ((b->l==0)&&(b->r==0))
            cout<<b->val<<"  ";
       else
           { if (b->l!=0) printLeaf(b->l);
             if (b->r!=0) printLeaf(b->r);
           }
     }
}
/*****************************************************************************/
void printPath(BTree b,int n)
{
  BTree t=b;
  cout<<"\nPath of "<<n<<" is : ";
  while ((t!=0)&&(t->val!=n))
    {
      cout<<t->val<<"  ";
      if (n<t->val) t=t->l;
      else          t=t->r;
    }
  if (t!=0) cout<<t->val<<endl;
  else      cout<<"The node "<<n<<" doesn't exist ! \n";
}
/*****************************************************************************/
int internalNum(BTree b)
{
  if ((b!=0)&&((b->l!=0)||(b->r!=0)))
       return 1+internalNum(b->l)+internalNum(b->r);
  else
       return 0;
}
```

```
/*****************************************************************************/
int leafNum(BTree b)
{
  if (b==0) return 0;
  else
      { if ((b->l==0)&&(b->r==0)) return 1;
        else return leafNum(b->l)+leafNum(b->r);
      }
}
/*****************************************************************************/
void treeHeight(BTree b,int h,int &maxH)
{ if ((b==0)||((b->l==0)&&(b->r==0)))
      { if (h>maxH) maxH=h; }
  else
      { treeHeight(b->l,h+1,maxH);
        treeHeight(b->r,h+1,maxH);
      }
}
/*****************************************************************************/
void fillLevels(BTree b,int a[],int i)
{ if (b!=0)
     { ++a[i];
       if (b->l!=0) fillLevels(b->l,a,i+1);
       if (b->r!=0) fillLevels(b->r,a,i+1);
     }
}
/*****************************************************************************/
int width(BTree b)
{ int H=0;
  treeHeight(b,0,H);
  for (i=0;i<=H;i++) a[i]=0;
  fillLevels(b,a,0);
  int wid=a[0];
  for (i=1;i<=H;i++)
       if (a[i]>wid) wid=a[i];
  return wid;
}
/*****************************************************************************/
int nodeHeight(BTree b,int x)
{ BTree t=b;
  int h=0;
  while ((t!=0)&&(t->val!=x))

    { ++h;
      if (x<t->val) t=t->l;
      else          t=t->r;
    }
  if (t!=0) return h;
  else      return 0;
}
/*****************************************************************************/
int LC(BTree b,BTree k)
{ if (k!=0) return nodeHeight(b,k->val)+LC(b,k->l)+LC(b,k->r);
  else      return 0;
}
/*****************************************************************************/
int LCI(BTree b,BTree k)
{ if ((k!=0)&&((k->l!=0)||(k->r!=0)))
       return nodeHeight(b,k->val)+LCI(b,k->l)+LCI(b,k->r);
  else return 0;
}
/*****************************************************************************/
```

```
int LCE(BTree b,BTree k)
{ if (k!=0)
       if ((k->l==0)&&(k->r==0)) return nodeHeight(b,k->val);
       else                      return LCE(b,k->l)+LCE(b,k->r);
  else                           return 0;
}
/*****************************************************************************************/
char isLinear(BTree b)
{ if (b==0) return 'y';
  else
  if ((b->l!=0)&&(b->r!=0)) return 'n';
  else
  if (b->l!=0) return isLinear(b->l);
  else         return isLinear(b->r);
}
/*****************************************************************************************/
char isComplete(BTree b)
{ int H=0;
  treeHeight(b,0,H);
  for (i=0;i<=H;i++) a[i]=0;
  fillLevels(b,a,0);
  char q='y';
  for (i=0;i<=H;i++)
      if (a[i]!=pow(2,i)) { q='n'; break; }
  return q;
}
/*****************************************************************************************/
char isPerfect(BTree b)
{
  Queue q;  emptyQueue(q);
  char c='n';
  if (b!=0) enqueue(q,b);
  while (isEmpty(q)=='n')
    {
      Top(q); dequeue(q);
      if (b==0) c='y';
      else
          if (c=='y') return 'n';
      if (b!=0)
         { enqueue(q,b->l);
           enqueue(q,b->r);
         }
    }
  return 'y';
}
```

تمرين :

اكتب خوارزمية تقوم بالتحقق من كون شجرتين متساويتين أم لا (ليس من الضرورة أن تكون الأشجار مرتبة !) .

الحل:

```
char equal(BTree b1,BTree b2)
{
  if ((b1==0)&&(b2==0)) return 'y';
  else
  if (((b1==0)&&(b2!=0))||((b1!=0)&&(b2==0)))   return 'n';
  else
  if (b1->val!=b2->val) return 'n';
  else
      { char q;
        q=equal(b1->l,b2->l);
        if (q=='y') q=equal(b1->r,b2->r);
        return q;
      }
}
```

تمرين :

اكتب خوارزمية حذف عقدة من شجرة بحث ثنائي .

الحل: يمكن إنشاء شجرة جديدة بحيث نقوم بنقل جميع عقد الشجرة الأصلية ما عدى العقدة المراد حذفها .

```
void deleteNode(BTree b,int x,BTree &b2)
{ if (b!=0)
     {
       if (b->val!=x) insertNode(b2,b->val);
       if (b->l!=0)   deleteNode(b->l,x,b2);
       if (b->r!=0)   deleteNode(b->r,x,b2);
     }
}
```

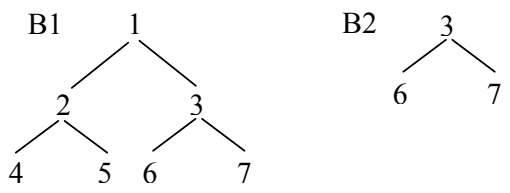
و من ثم نقوم بإسناد الشجرة الجديدة b2 إلى القديمة b .

تمرين:

نقول عن شجرة ثنائية B2 إنها جزئية من شجرة ثنائية أخرى B1 إذا كانت B1 تحوي كامل B2 (كما في الشكل التالي).المطلوب:

1- اكتب إجرائية تتحقق من كون شجرة B2 جزئية من شجرة أخرى B1 .

2- اكتب إجرائية تقوم بحساب تكرار وجود B2 في B1 .



الحل: يمكن دمج الطلبين بكتابة الخوارزميتين التاليتين :

```
char equal(BTree b1,BTree b2)
{
  if (b2==0) return 'y';
  else
  if ((b1==0)&&(b2!=0)) return 'n';
  else
  if ((b1->val!=b2->val)) return 'n';
  else
      {
        char q=equal(b1->l,b2->l);
```

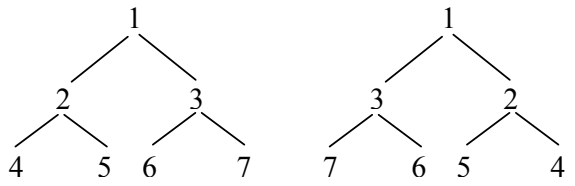
```
        if (q=='y') q=equal(b1->r,b2->r);
        return q;
      }
}
/*************************************************************************************/
void frequency(BTree b1,BTree b2,int &n)     //  عدد تكرار الوجود
{
  if (b1!=0)
     {
       if (equal(b1,b2)=='y') ++n;
       frequency(b1->l,b2,n);
       frequency(b1->r,b2,n);
     }
}
/*************************************************************************************/
```

حيث نستدعي في البرنامج الرئيسي كالتالي :

```
  int n=0;
  frequency(b1,b2,n);
  if (n>0) cout<<"b2 is part of b1 "<<n<<" times ";
  else     cout<<"b2 is not part of b1 ";
```

تمرين:

نقول عن شجرتين ثنائيتين أنهما متشابهتان بالتعاكس (Mirror Similar) إذا كانتا فارغتين معاً , أو إذا كانتا غير فارغتين و الشجرة الجزئية اليسارية من كلتيهما متشابهة بالتعاكس مع الشجرة الجزئية اليمينية من كلتيهما (كما في الشكل التالي) . اكتب خوارزمية تتحقق من كون شجرتين ثنائيتين , متشابهتين بالتعاكس .



الحل:

```
char isMirror(BTree b1,BTree b2)
{
  if ((b1==0)&&(b2==0)) return 'y';
  else
  if (((b1==0)&&(b2!=0))||((b1!=0)&&(b2==0))) return 'n';
  else
  if (b1->val!=b2->val) return 'n';
  else
      { char q;
        q=isMirror(b1->l,b2->r);
        if (q=='y') q=isMirror(b1->r,b2->l);
        return q;
      }
}
```

الأشجار المعممة – Generalized Trees

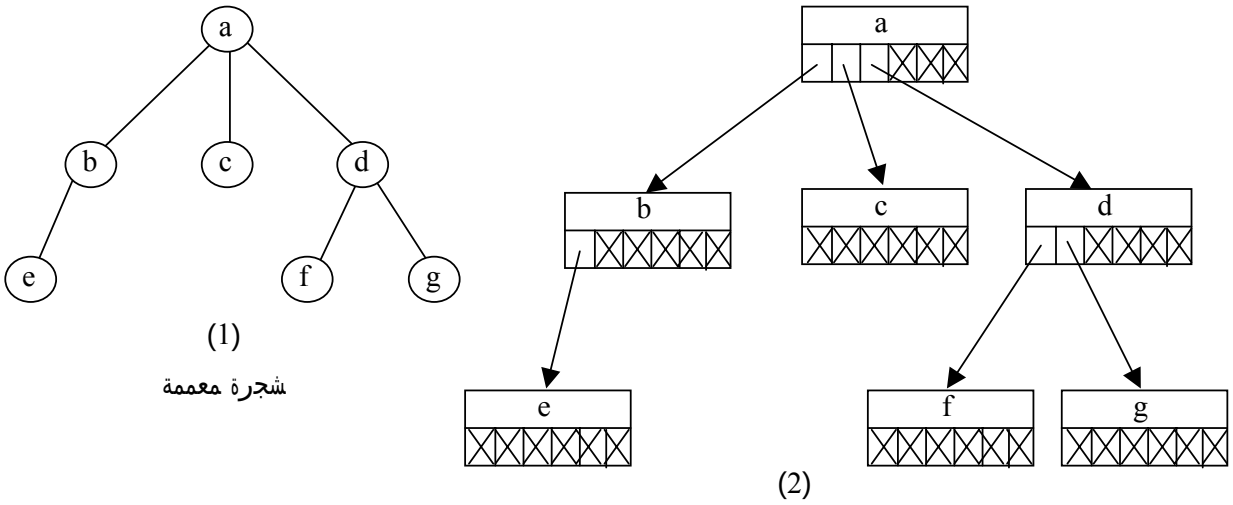
الشجرة المعممة هي شجرة يمكن فيها وجود عدة أبناء لكل عقدة. يمكن تمثيل الأشجار المعممة بطرق عدة , نعرض منها الطريقتين التاليتين :

1- تمثيل الشجرة المعممة باستعمال مصفوفة من المؤشرات :

تحوي العقدة الواحدة مصفوفة من المؤشرات للدلالة على الأولاد , و يعرف عدد أعظمي للمؤشرات (= درجة الشجرة) التي تصدر عن كل عقدة. بالطبع نلاحظ بأن هذا التمثيل يسبب هدر في الذاكرة المحجوزة, ثم إن هذا العدد الأعظم للأولاد قد يكون غير كاف في بعض الحالات.

مثال:

لتكن الشجرة المعممة المبينة في الشكل (1) , إذا اعتبرنا أن درجة الشجرة (العدد الأعظم للأولاد) هي 6 , يمكن تمثيل الشجرة باستعمال مصفوفة من المؤشرات , كما في الشكل (2) :



تمثيل شجرة معممة باستخدام مصفوفة مؤشرات

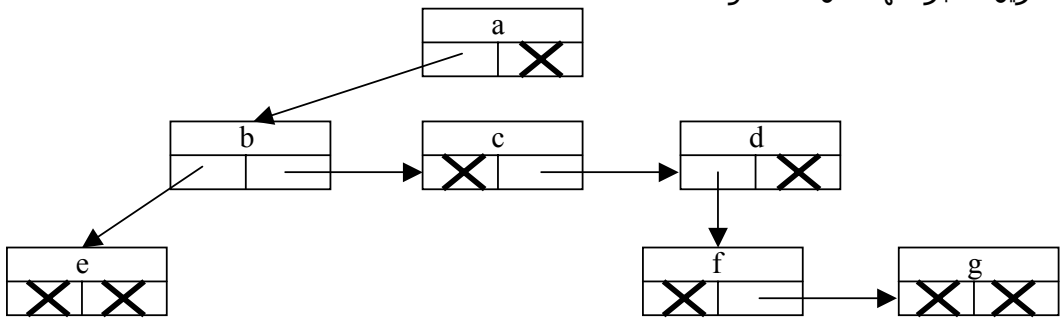
حيث يتم هنا تعريف بنية الشجرة المعممة بالشكل التالي :

```
const n=6;      //  n = Degree of GTree
struct gnode{
       element val;    // int , char , float , ...
       gnode* a[n];
};
typedef gnode* GTree;
```

2- تمثيل الشجرة المعممة باستعمال سلاسل من الأولاد :

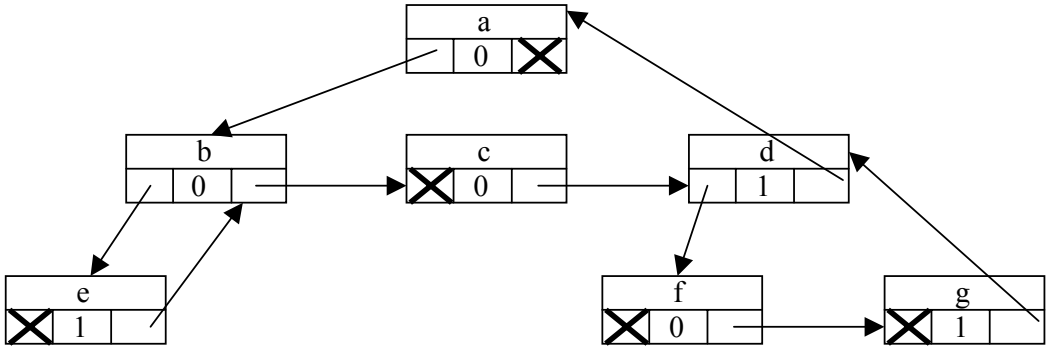
تحوي العقدة مؤشرين , يدل الأول على الولد الأول و يدل المؤشر الثاني على الأخ التالي. في هذه الحالة , لا يوجد ضياع في الذاكرة المحجوزة , و يمكن تخزين الشجرة مهما كان عدد الأولاد .

مثال:



يمكن تعديل هذه الطريقة في التمثيل بجعل آخر ولد يؤشر إلى الأب , و من أجل ذلك نحتاج إلى حقل إضافي لنحدد العقدة التالية : أهي أخ أو أب ؟ إذاً لنضيف حقل من نوع عدد صحيح , إذا كانت قيمته "0" فهذا يعني أن العقدة تؤشر إلى أخ , و إذا كانت قيمته "1" فهذا يعني أن العقدة تؤشر إلى الأب .

مثال:

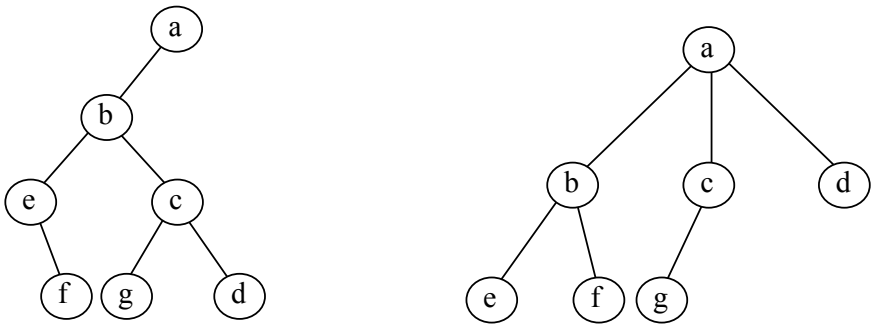


❖ تحويل شجرة معممة إلى شجرة ثنائية :

إذا لم يكن هناك ترتيب معرف على أبناء العقد في شجرة معممة , فيمكن تحويلها إلى شجرة ثنائية بواسطة الخوارزمية التالية :

- الابن الأول لعقدة في الشجرة المعممة يصبح الابن اليساري لنفس العقدة في الشجرة الثنائية .
- أخو عقدة في الشجرة المعممة يصبح الابن اليميني لنفس العقدة في الشجرة الثنائية .

مثال:



شجرة معممة — شجرة ثنائية مكافئة

تمرين:

اكتب خوارزمية لطباعة الأوراق في شجرة معممة g , حيث تم تمثيل g باستخدام مصفوفات من المؤشرات.

الحل:

```
void printLeafs(Gtree g)
{ if (g!=0)
      if (g->a[0]==0)
          cout<<g->val;
      else
          for (int i=0;i<n;i++)
              printLeafs(g->a[i]);
}
```

تمرين:

اكتب خوارزمية التحويل من شجرة معممة إلى شجرة ثنائية .

الحل:

const n=6;

```
struct node{
       int val;
       node *l,*r;
};
typedef node* BTree;

struct gnode{
       int val;
       gnode* a[n];
};
typedef gnode* GTree;

BTree b=0; GTree g=0;
/*****************************************************************************************/
void c(GTree g,BTree &b)
{
  if (g->a[0]!=0)
     {
       b->l=new node();
       b->l->val=g->a[0]->val;
       b->l->l=0; b->l->r=0;
     }
  else  return ;  //  إنهاء الإجراء
  BTree b1=b->l;
  for (int i=1;i<n;i++)
      if (g->a[i]!=0)
         {
           b1->r=new node();
           b1=b1->r;
           b1->val=g->a[i]->val;
           b1->l=0; b1->r=0;
         }
      else
            break;
  b1=b->l;
  for (i=0;i<n;i++)
      if (g->a[i]!=0)
         {
           c(g->a[i],b1);
           b1=b1->r;
         }
      else
           break;
}
/*****************************************************************************************/
void convert(GTree g,BTree &b)
{ if (g!=0)
     {
       b=new node();
       b->val=g->val;
       b->l=0; b->r=0;
       c(g,b);
     }
}
```

Graphs – البيانات

إن البيانات عبارة عن تعميماً للبنى غير التسلسلية من الناحية النظرية , حيث أن الأشجار حالة خاصة من البيانات .

تعريف البيان - Graph :

هو ثنائية $G=<V,E>$ حيث V مجموعة منتهية من **العقد** (vertices) و E مجموعة منتهية من الثنائيات المرتبة $<v1,v2>$ حيث $v1,v2 \in V$. تسمى هذه الثنائيات **أسهما** (أو حافات edges) .

نقول عن عقدتان أنهما **متجاورتان** إذا وجد سهم بينهما .

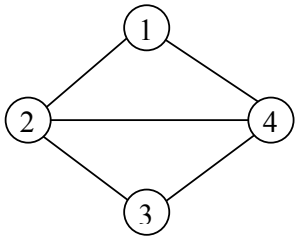
يوجد نوعين من البيانات :

1- **البيانات غير الموجهة (Undirected Graph) :** تكون فيها الثنائيات التي تمثل الأسهم غير مرتبة: $<v,w>=<w,v>$

2- **البيانات الموجهة (Directed Graph) :** تكون فيها الثنائيات التي تمثل الأسهم مرتبة , أي : $<v,w>\neq<w,v>$

ملاحظات:

- إذا كان $<v,w>$ سهماً في البيان فإن : $v\neq w$.
- بين عقدتين يمكن أن يوجد سهمان موجهان على الأكثر في حال كون البيان موجهاً , و سهماً وحيداً على الأكثر في حال كون البيان غير موجه .



مثال 1 :

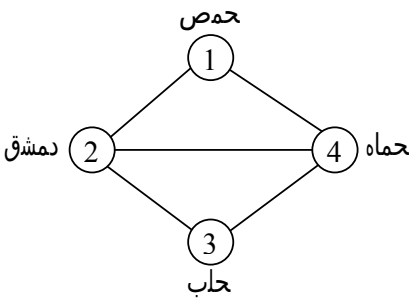
ليكن البيان غير الموجه التالي G :

$G = <V,E>$

$V<G> = \{1,2,3,4\}$

$E(G) = \{<1,2>,<1,4>,<2,3>,<2,4>,<3,4>\}$

كمثال عملي يمكن أن تمثل كل عقدة محافظة ما و بالتالي لتكون الأسهم عبارة عن خطوط النقل الممكنة ما بين المحافظات (علماً بأن خط نقل باتجاهين , أي إذا وجد خط ما بين محافظة أولى إلى ثانية بالطبيعة يوجد خط من المحافظة الثانية إلى الأولى) أي:



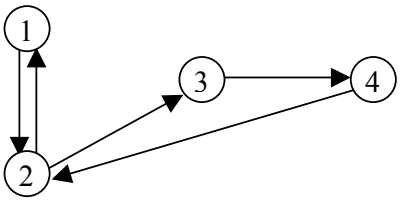
مثال 2 :

ليكن البيان الموجه التالي G2 :

$G2 = <V,E>$

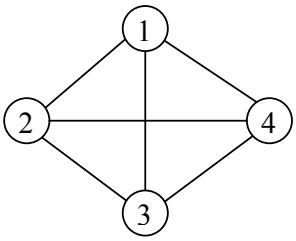
$V<G2> = \{1,2,3,4\}$

$E(G2) = \{<1,2>,<2,1>,<2,3>,<3,4>,<4,2>\}$



ملاحظة: عدد الروابط الأعظم في بيان غير موجه يحوي n عقدة يساوي $\frac{n(n-1)}{2}$, و بالتالي نقول عن البيان أنه **بيان تام** (**complete graph**) .

مثال 3 :



ليكن البيان غير الموجه و التام التالي : G

```
G = <V,E>
V<G> = {1,2,3,4}
E(G) = {<1,2>,<1,3>,<1,4>,<2,3>,<2,4>,<3,4>}
```

تعريف البيان الجزئي :

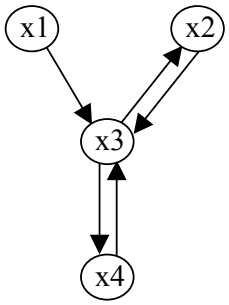
هو ثنائية $<V', E'>$ حيث :

- $V' \subseteq V$ and $E' \subseteq E$
- $<v,w> \subseteq E' \Leftrightarrow v \in V'$ and $w \in V'$

تعريف درجة العقدة :

درجة العقدة Deg(v) هي عدد الأسهم الواردة إلى هذه العقدة و الصادرة عنها . لذلك نميز ما بين الدرجة الواردة $\overline{Deg}(v)$ و الدرجة الصادرة $\overset{+}{Deg}(v)$.

مثال:



ليكن لدينا البيان الموجه التالي

Deg(x3) = 5	Deg(x4) = 2	Deg(x1) = 1
$\overset{+}{Deg}$(x3) = 2	$\overset{+}{Deg}$(x4) =1	$\overset{+}{Deg}$(x1) = 1
$\overline{Deg}$(x3) = 3	$\overline{Deg}$(x4) = 1	$\overline{Deg}$(x1) = 0

تعريف الطريق (path) :

الطريق ضمن بيان هو كل متتالية $v_i , v_{i+1} , ... , v_j$ من العقد , حيث يوجد أسهم بين v_k و v_{k+1} عندما $i \le k \le j-1$.

و يكون الطريق **بسيطاً (elementary)** إذا لم تتكرر أي عقدة فيه مرتين .

تعريف الحلقة (cycle) :

هي طريق بسيط بحيث $v_i = v_j$, و نسميها **حلقة موجهة** في حالة البيان موجه .

تعريف البيان متصل بقوة (strongly connected) :

لكل عقدتين v و w من البيان يوجد طريق من v إلى w , و طريق من w إلى v

❖ **تمثيل البيانات الموجهة :**

يمكن تمثيل بيان موجه بطرق عديدة حيث نذكر منها ثلاثة :

1- تمثيل البيان الموجه باستعمال مصفوفة تجاور (مصفوفة من المتحولات المنطقية) :

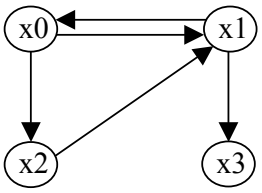
تكون مصفوفة التجاور (Adjacency Matrix) M في هذه الحالة مربعة , الأسطر و الأعمدة فيها هي أرقام العقد . نضع في العنصر M[i][j] القيمة true (أو "y") إذا كان يوجد رابط موجه من العقدة v_i إلى العقدة v_j . على حين نضع القيمة false (أو "n") عند عدم وجود رابط بين العقدتين . نلاحظ بأن في حالة البيان غير الموجه , تصبح هذه المصفوفة متناظرة بالنسبة إلى القطر الرئيسي .

إذاً يصبح التمثيل على الشكل التالي:

```
const n=100;
typedef char MGraph[n][n];
```

مثال :

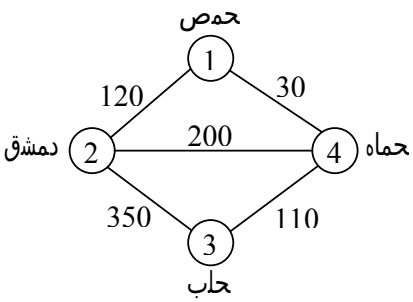
يتم تمثيل البيان الموجه المجاور بالشكل التالي :



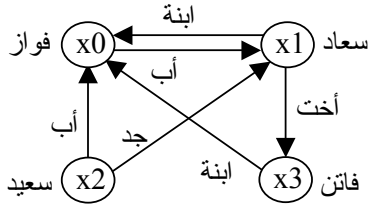
n	Y	Y	n
Y	n	n	Y
n	Y	n	n
n	n	n	n

2- تمثيل البيان الموجه باستعمال مصفوفة تعطي كلفة قوس :

يمكن للبيان الموجه أن يبين كلفة الانتقال من عقدة إلى أخرى , فمثلاً إذا عدنا إلى مثال المحافظات و خطوط النقل الممكنة ما بينها فيمكن إضافة البعد ما بين المحافظات , أو سعر بطاقة النقل , أو الزمن اللازم للانتقال , ... الخ . لنأخذ كمثال المسافات بالكيلومتر .



لنأخذ مثال آخر : يمكن أن نبين نوع القرابة ما بين عدة أشخاص :

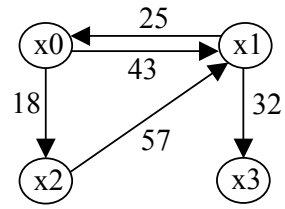


تكون مصفوفة التجاور M في هذه الحالة أيضاً مربعة , الأسطر و الأعمدة فيها هي أرقام العقد . نضع في العنصر M[i][j] قيمة كلفة الانتقال من العقدة v_i إلى العقدة v_j , إذا كان يوجد رابط موجه من العقدة v_i إلى العقدة v_j . على حين نضع القيمة -1 عند عدم وجود رابط بين العقدتين . نلاحظ أيضاً أنه في حالة البيان غير الموجه , تصبح هذه المصفوفة متناظرة بالنسبة إلى القطر الرئيسي . إذاً يصبح التمثيل على الشكل التالي:

```
const n=100;
typedef int MVGraph[n][n];      //  or: float , char , ...   حسب نوع الكلفة
```

مثال :

يتم تمثيل البيان الموجه المجاور بالشكل التالي:



-1	43	18	-1
25	-1	-1	32
-1	57	-1	-1
-1	-1	-1	-1

3- تمثيل البيان الموجه بواسطة سلاسل التجاور :

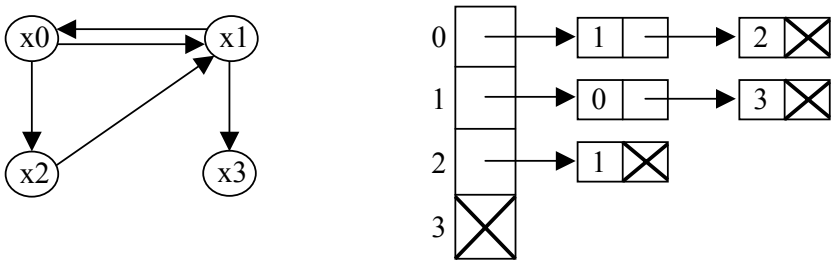
في حالة تمثيل البيان بواسطة سلاسل التجاور (Adjacency Lists) , نمثل البيان بواسطة شعاع من المؤشرات بعده هو عدد العقد في البيان , و مرقم حسب أرقام العقد . خانة كل عقدة i هي مؤشر إلى بداية سلسلة خطية تحوي جميع العقد j التي ترتبط بـ i وفق الاتجاه i إلى j , و تكون هذه السلسلة مرتبة تصاعدياً حسب أرقام العقد .

إذاً يصبح التمثيل على الشكل التالي:

```
const n=100;
struct vertex{
       element val;    //  element= int , float , char , ...
       vertex* next;
};
typedef vertex* LGraph[n];
```

مثال :

يتم تمثيل البيان الموجه المجاور بواسطة سلاسل التجاور على الشكل التالي:



❖ **بعض العمليات على البيانات :**

يوجد عدد كبير من العمليات المهمة التي تجري على البيانات , نختار منها العمليات التالية :

مسح البيانات - Graph Traversal

1- مسح البيان غير الموجه :

بفرض $G = < V , E >$ بيان غير موجه و v عقدة ما منه . فمسح هذا البيان (Graph Traversal) يعني زيارة جميع العقد في G التي يمكن الوصول إليها بطريق من v . هناك طريقتين للمسح : العمق أولاً و العرض أولاً .

1-1- مسح البيان بالعمق أولاً - Depth First :

يتم مسح عقدة البداية أولاً (حيث يمكن تحديدها حسب الرغبة) , ثم نختار عقدة مجاورة لها غير ممسوحة بعد , ونتابع هكذا بطريقة عودية عند هذه العقدة ... , عندما نصل إلى عقدة ليس لها عقد مجاورة غير ممسوحة , نعود إلى آخر عقدة ممسوحة و لها عقد مجاورة لم تمسح بعد . ننتهي عند مسح جميع العقد .

بفرض $G = < V , E >$ بيان غير موجه و v عقدة البداية , لنمثل العقد الممسوحة بالشعاع التالي:

```
char visited[n];
```

حيث نعطي قيم بدائية لجميع عناصر هذا الصف 'n' , أي جميع العقد غير ممسوحة , و نغير المحتوى إلى 'y' عند مسح العقدة.

لنكتب خوارزمية المسح بالعمق أولاً بلغة الـ pseudocode :

```
char visited[n];
for (int i=0;i<n;i++) visited[i]='n';
void DFS(int v)          // v = دليل عقدة البداية
{
  int w;                 // w = دليل عقدة ما
  visited[v]='y';
  for (each vertex w adjacent to v)
      if (visited[w]=='n')
          DFS(w);
}
```

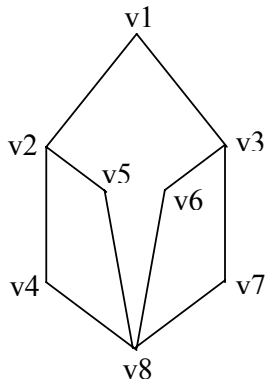
2-1- مسح البيان بالعرض أولاً - Breadth First :

يتم مسح عقدة البداية أولاً , ثم نمسح جميع العقد المجاورة لها غير الممسوحة بعد و نضعها في رتل (Queue). من أجل كل عقدة متبقية في الرتل نمسح العقد المجاورة لها و نضعها في رتل و هكذا . ننتهي عندما يفرغ الرتل كاملةً .

تصبح خوارزمية المسح بالعرض أولاً بلغة الـ pseudocode على الشكل التالي :

```
void BFS(int v)          // v = دليل عقدة البداية
{
  char visited[n];
  int w;                 // w = دليل عقدة ما
  Queue q;
  for (int i=0;i<n;i++) visited[i]='n';
  visited[v]='y';
  emptyQueue(q);
  do{
      for (all vertices w adjacent to v)
          if (visited[w]=='n')
             {
               enqueue(q,w);
               visited[w]='y';
             }
      if (isEmpty(q)=='y') exit;
      else
          {
            v=Top(q);
            dequeue(q);
          }
  } while (1==1);  //  exit حلقة لانهائية لا تنتهي إلا إذا تم تفريغ الرتل و بالتالي نخرج بتعليمة
}
```

مثال : ليكن لدينا البيان غير الموجه التالي و لتكن v1 عقدة البداية :



نتيجة مسح البيان بالعمق أولاً (=الطواف بالعمق) :

v1,v2,v4,v8,v5,v6,v3,v7

نتيجة مسح البيان بالعرض أولاً (=الطواف بالعرض) :

v1,v2,v3,v4,v5,v6,v7,v8

2- مسح البيان الموجه :

لا تختلف طرق المسح و إجرائياتها هنا عن تلك الإجرائيات المطبقة في مسح البيان غير الموجه . الفرق الوحيد هو مراعاة اتجاه الأسهم . لذلك عندما يجب أن تمسح الخوارزمية العقدة j المجاورة للعقدة i التي عندها الخوارزمية , يجب أن يكون السهم بين i و j بالاتجاه $i > j$, بمعنى آخر يجب أن تنتمي العقدة j إلى $Deg^{+}(i)$ (و التي نعتبرها هنا مجموعة العقد المجاورة لـ i في حالة بيان موجه).

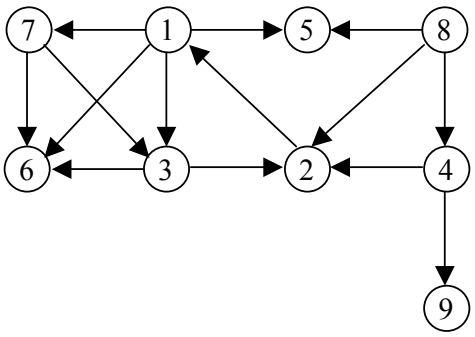
لنأخذ طريقة العمق أولاً و نعيد كتابة الخوارزمية DFS , حيث نستخدم هنا الإجرائية المساعدة OrDFS التي تسمح بالمرور بجميع العقد التي يمكن الوصول إليها من عقدة معينة و التي لم يسبق المرور فيها .

(pseudocode)

```
void DTraversal(Graph G)
{
  char visited[n];
  for (int i=0;i<n;i++) visited[i]='n';
  for (i=0;i<n;i++)
      if (visited[i]=='n')
         DFS(i,visited);
}
```

```
void DFS(int v,char visited[])
{
  visited[v]='y';
  for (int j=0;j< Deg+(v);j++)
      if (visited[j]=='n')
          DFS(j,visited);
}
```

مثال:



عند تطبيق خوارزمية التجول بالعمق أولاً على البيان G الموضح في الشكل المجاور, فإن ترتيب المرور في العقد هو:

1 , 3 , 2 , 6 , 5 , 7 ,
4 , 9 ,
8

أما بالنسبة للتجول بالعرض أولاً يكون ترتيب المرور :

1 , 3 , 5 , 6 , 7 ,
2 ,
4 , 9 ,
8

تمرين :

ليكن G بيان موجه تم تمثيله باستخدام سلاسل التجاور . اكتب خوارزمية تختبر إن كان هذا البيان عبارة عن شجرة ثنائية أم لا .

الحل: يجب أن يكون لكل عقدة في البيان عقدتان متجاورتان على الأكثر لأن نعتبر البيان عبارة عن شجرة ثنائية , إذاً :

```
const n=10;
srtuct vertex{
       int val;
       vertex* next;
};
typedef vertex* LGraph[n];
LGraph G;
/*****************************************************************************************/
char isBTree(LGraph G)
{
  int sum=0;     //  عداد يعد عدد العقد المجاورة لكل عقدة
  vertex *p,*q;  //  مؤشرين مساعدين إلى عقدة
  for (int i=0;i<n;i++)
      {
        p=G[i];
        sum=0;
        while (p!=0)
          { ++sum;
            if (sum>2) return 'n';
            for (j=0;j<n;j++)
                if (j<>i)
                   {
                     q=G[j];
                     while (q!=0)
                       {
                         if (q->val==p->val) return 'n';
                         q=q->next;
                       }
                   }
            p=p->next;
          }
      }
  return 'y';
}
```